Techniques de contrôle du mouvement pour l'animation

Gabriel Hanotaux

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

UNIVERSITE JEAN MONNET
DE SAINT ETIENNE

N° d'ordre : 86 ID

THESE

présentée par

Gabriel HANOTAUX

pour obtenir le titre de

DOCTEUR EN INFORMATIQUE

DE L'UNIVERSITE DE SAINT-ETIENNE

ET DE L'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

(Spécialité : Image)

Techniques de contrôle du mouvement pour l'animation

Soutenue à Saint-Etienne le 22 avril 1993

COMPOSITION DU JURY

Monsieur	PEROCHE	Président
Messieurs	HEGRON	Rapporteurs
	THALMANN	
Madame	GASCUEL	Examinateurs
Messieurs	AZEMA	
	GARDAN	

ECOLE NATIONALE SUPERIEURE
DES MINES DE SAINT-ETIENNE

UNIVERSITE JEAN MONNET
DE SAINT ETIENNE

Nº d'ordre : 86 ID

THESE

présentée par

Gabriel HANOTAUX

pour obtenir le titre de

DOCTEUR EN INFORMATIQUE

DE L'UNIVERSITE DE SAINT-ETIENNE

ET DE L'ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

(Spécialité : Image)

Techniques de contrôle du mouvement pour l'animation

Soutenue à Saint-Etienne le 22 avril 1993

COMPOSITION DU JURY

Monsieur	PEROCHE	Président
Messieurs	HEGRON	Rapporteurs
	THALMANN	
Madame	GASCUEL	Examinateurs
Messieurs	AZEMA	
	GARDAN	

Remerciements

Je tiens ici à exprimer mes remerciements aux personnes qui ont accepté de juger ce travail ainsi qu'à celles qui m'ont aidé au long de cette thèse.
Merci à Gérard Hégron et Daniel Thalmann d'avoir bien voulu être rapporteurs de ce travail, à Jean Azéma et Yvon Gardan pour leur présence dans ce jury, à Marie-Paule Gascuel pour le temps qu'elle a consacré à relire ce manuscrit et les améliorations qu'elle a suggérées et à Bernard Péroche pour m'avoir accueilli dans son laboratoire et conseillé dans mes travaux.
Je tiens à exprimer mon amitié à l'ensemble du département informatique de l'Ecole des Mines de Saint-Etienne, Marie Line, Philippe, Marc, Jean Luc toujours au rendez-vous de la machine à café, Roland avec qui j'ai partagé mon bureau lors de ce passage aux Mines, Jean Michel, Annie, Elizabeth, Florence, Mohand, Véronique, François, Gilles, Dominique ainsi que tous les autres qui se reconnaitront.
Je n'oublie pas le personnel du service de reprographie qui a assuré la réalisation de ce rapport.
Je terminerai avec un grand merci à mes parents pour leur soutien tout au long de mes études et à Pascale pour tout.

Sommaire

Introduction

Recréer l'univers qui nous entoure; telle est la quête que poursuit depuis quelques décennies le monde de l'image de synthèse. Avec les progrès accomplis récemment, le fossé séparant il y seulement dix ans la réalité et les images synthétiques se comble rapidement. Des phénomènes naturels de plus en plus complexes sont simulés grâce à des modèles approchant toujours plus finement la réalité.

Donner vie au monde figé des images de synthèse est le domaine de l'animation par ordinateur. Là aussi, la recherche du réalisme hante les chercheurs. Une preuve en est fournie par le nombre sans cesse croissant d'apparitions du mot simulation dans les papiers traitant d'animation. Recréer artificiellement le mouvement humain est l'une des gageures de l'animation par ordinateur. Cependant l'animation par ordinateur ne se contente pas de simuler la réalité. Un autre aspect intéressant est de permettre la création de mouvements que l'on ne rencontre pas dans la réalité, et cela en passant par une description détaillée du mouvement. Il est possible aussi d'amplifier certaines caractéristiques afin de renforcer les effets visuels ou de donner au mouvement un effet comique. On rejoint là les techniques utilisées dans le domaine de l'animation traditionnelle, comme par exemple l'écrasement exagéré d'une balle rebondissant par terre.

La création d'images animées se décompose globalement en trois étapes principales qui sont la modélisation, la génération du mouvement et la visualisation :

- La première de ces étapes consiste à définir la représentation des différents objets intervenant dans l'animation. Le choix des techniques permettant de définir la géométrie de ces objets est vaste; des arbres de construction aux représentations par frontières en passant pas les arbres octaux et autres BSP. Cette représentation doit permettre de créer des images des objets ainsi modélisés. Elle est donc fortement dépendante de l'algorithme de visualisation utilisé. Bien que l'on associe habituellement animation et mouvement, l'animation par ordinateur est aussi concernée par tout ce qui peut avoir un effet visuel. D'autres caractéristiques, qui peuvent ou non être animées, telles que la couleur, l'aspect, etc..., doivent aussi être définies.

  Par ailleurs, la représentation informatique des objets d'une scène doit tenir compte des techniques d'animation employées. Souvent les informations énumérées ci-dessus suffisent. Quand, par exemple, il s'agit d'appliquer des principes issus de la mécanique des solides, d'autres informations telles que la masse, des coefficients d'élasticité,

etc...sont nécessaires.

- La partie animation consiste à déterminer les valeurs que prennent l'ensemble des paramètres définissant les objets de la scène au cours du temps. Là aussi, les techniques sont multiples. Elles dépendent des capacités de l'animateur, de l'effet recherché ou du réalisme désiré.

- Enfin, la dernière étape consiste à générer l'ensemble des images, qui mises bout à bout et affichées à la vitesse adéquate, formeront l'animation.

  Pour préciser le cadre dans lequel se sont effectués nos travaux, nous utilisons un système de synthèse d'image basé sur un modeleur par arbre de construction [BP90]. L'affichage, c'est à dire l'élimination des parties cachées est réalisé soit par l'algorithme d'Atherton (un algorithme par balayage de lignes – style Watkins – résolvant les opérations booléennes) soit par tracé de rayon.

Pour en revenir à la partie purement animation, on dénombre de nombreuses approches : animation par paramètres-clés, animation programmée, animation par scripts et acteurs, animation dirigée par les buts, animation dynamique plus une multitude de techniques spécialisées (voir par exemple [Gan89] [Arn88]). Certaines privilégient le réalisme, d'autres la possibilité de contrôle du mouvement, certaines s'adressent à un utilisateur informaticien, d'autres non.

Afin de situer les travaux présentés, nous adopterons une classification plus large qui, aux termes de vocabulaire près, est bien définie. On distingue principalement deux grandes catégories : les modèles descriptifs et les modèles générateurs, aussi appelés modèles phénoménologiques et modèles causaux.

Les premiers s'attachent à décrire les mouvements de manière explicite. Il s'agit de définir les caractéristiques des objets au cours du temps, traité explicitement, à l'aide de fonctions d'évolution. Celles-ci peuvent être des fonctions mathématiques décrivant par exemple la trajectoire spatiale d'un objet au cours du temps. Plus couramment, les paramètres des objets sont définis à des instants particulièrement représentatifs de l'animation. La tâche fondamentale du système d'animation est alors de calculer les valeurs intermédiaires prises par l'ensemble des paramètres. Pour cela, des méthodes d'interpolation bien connues sont utilisées. Essentiellement, le mouvement est défini par les valeurs des paramètres à des instants spécifiques, par l'allure des courbes d'interpolation mais aussi par la vitesse d'évolution des paramètres au cours du temps.

Les modèles générateurs quant à eux s'intéressent aux causes qui engendrent le mouvement. Une distinction est généralement faite suivant que les objets à animer sont inertes (une balle lancée en l'air) ou au contraire ont un comportement spécifique. Contrairement aux modèles descriptifs, le temps est traité de manière implicite; par exemple au travers des équations différentielles traduisant les lois de la mécanique. Le contrôle du mouvement dépend ici des lois que subissent les objets ou des comportements qu'ils doivent respecter.

Dans cette thèse, constituée de deux parties, nous aborderons les deux points de vue.

La première partie concerne les modèles descriptifs. Nous nous intéresserons plus particulièrement au contrôle interactif des animations par paramètres-clés. Un objectif est spécialement visé: fournir des solutions interactives performantes pour la description des interpolations, plus précisemment, assurer la manipulation en temps réel des trajectoires décrivant l'animation des positions mais aussi des orientations. Les problèmes de l'interpolation des orientations, de l'interface, de la vitesse le long des trajectoires seront abordés.

Dans la deuxième partie seront présentées quelques unes des approches les plus récentes et qui, de notre point de vue, font la liaison entre modèles descriptifs et modèles générateurs. Ces approches peuvent être être considérées comme faisant partie des modèles descriptifs puisque l'animateur spécifie les valeurs de certains paramètres à des instants spécifiques. Elles peuvent aussi être classées parmi les modèles générateurs puisque l'interpolation entre les paramètres respectent des lois – en particulier, les lois de la mécanique des solides – propres aux modèles générateurs.

Ces approches utilisent le fait qu'un mouvement réel tend à consommer le minimum d'énergie. Elles utilisent soit des techniques de contrôle optimal pour des équations linéaires du mouvement, soit des techniques d'optimisation pour des équations non linéaires discrétisées, soit des techniques de la programmation dynamiques et sont en général très gourmandes en temps et place mémoire. Nous présentons une nouvelle approche, basée sur le calcul des variations, où les équations du mouvement sont des équations non linéaires continues.

Première partie

Animation paramétrée

Introduction

D'un point de vue historique, l'animation basée sur des modèles descriptifs est à l'origine de l'animation par ordinateur. L'objectif initial était d'automatiser les systèmes traditionnels de production d'images animées. L'animation par ordinateur s'est donc inspirée, au départ, des techniques de l'animation traditionnelle.

L'ordinateur est particulièrement adapté à l'automatisation de certaines étapes de la création d'animations traditionnelles. Certaines images, particulièrement caractéristiques du mouvement, sont dessinées à la main par l'animateur, les images intermédiaires étant calculées par interpolation. Un tel système se compose en général d'un éditeur interactif permettant de définir les images clés, d'en marquer certains points caractéristiques et de techniques d'interpolations. Le principe de ces systèmes à base d'images clés est à l'origine de nombreuses méthodes actuelles d'animation .

La plus simple des méthodes d'interpolation est l'interpolation linéaire. Etant donnés deux valeurs v_d et v_f d'un attribut d'un objet aux instants initiaux et finaux, les valeurs intermédiaires sont calculées à partir de l'équation bien connue $v_t = (1-t)v_d + tv_f$ quand t varie entre 0 et 1. Cependant, l'interpolation linéaire, en dehors de cas particuliers souffre de nombreuses limitations. Ce type d'interpolation génére un mouvement continu; par contre les dérivées le sont rarement ce qui entraîne des variations de vitesse désagréables et par conséquent fort peu réalistes. Par exemple, simuler la trajectoire d'un balle soumise à la gravité est impossible avec une interpolation linéaire (figure 0.1).

Pour résoudre ces problèmes, les courbes et fonctions splines ont été utilisées pour interpoler entre des valeurs clés. Elles assurent la continuité des dérivées, c'est à dire des trajectoires lisses.

Un autre aspect important concerne le choix des paramètres. Paramétrer un objet consiste à fournir un ensemble de caractéristiques permettant d'afficher l'objet. Un objet

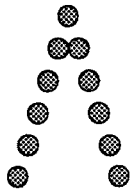


FIG. 0.1 Problème de l'interpolation linéaire.

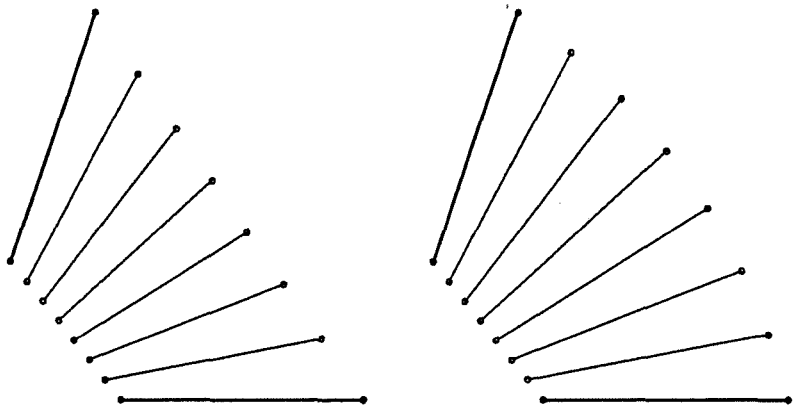


FIG. 0.2 Deux animations différentes obtenues en interpolant deux types de paramétrage: (a) extrémités du segment (b) une extrémité et un angle.

peut en effet être paramétré de multiples manières.

Un exemple fréquemment cité est celui du segment de droite. Il peut être représenté soit par les coordonnées de ses deux extrémités, soit par la donnée de la position d'une extrémité, d'un angle et de sa longueur (constante). Les images intermédiaires obtenues en interpolant le premier jeu de paramètres fournit des segments intermédiaires de longueur variable (figure 0.2.a). Au contraire, l'interpolation du deuxième type de paramétrage (position et angle) conduit à une interpolation cohérente (figure 0.2.b). Le premier type d'interpolation est utilisé dans les systèmes d'animation à base d'images-clés. Les objets constituant l'image sont souvent composés de figures géométriques et l'interpolateur se charge de calculer les positions intermédiaires des sommets de ces figures.

Dans le cas des systèmes d'animation tri-dimensionnels, les objets sont modélisés explicitement. L'animateur n'a pas besoin de décrire l'évolution des points constituant l'objet à animer. On paramétrise généralement un objet par une position et une orientation. Ces deux paramètres définissent le repère local de l'objet. La géométrie de l'objet est définie dans ce repère local.

Les fonctionnalités que se doit d'assurer un système d'animation par paramètres-clés sont principalement :

- la définition des paramètres à chaque position-clé,
- l'interpolation des paramètres au cours du mouvement. Celle-ci doit tenir compte de la nature des paramètres,
- le spécification des vitesses au cours du mouvement

Actuellement, la majorité des travaux ont été réalisés sur les trajectoires interpolant les positions. Les différents points énumérés ci-dessus seront abordés au cours de cette

première partie, avec une attention particulière au cas des orientations.

Dans le premier chapitre, nous nous intéresserons au contrôle interactif des orientations d'un objet en mouvement. L'objectif est de fournir une méthodologie comparable à ce qui existe pour l'interpolation des positions. La paramétrisation des orientations par les logarithmes de quaternion et la notion de tangente sphériques permettrons d'atteindre ce but, c'est à dire un contrôle en temps réel des interpolations d'orientations.

Dans le deuxième chapitre, nous décrirons le système d'interface qui a été utilisé. Les fonctionnalités de bas niveau, assurant la mise en correspondance entre les mouvements de la souris et les déplacements sur la scène 3-D seront détaillées. Nous proposerons aussi un algorithme de cinématique inverse particulièrement adapté à un environnement interactif.

En général, l'animation par paramètres-clés requiert de la part de l'utilisateur la spécification de nombreuses courbes d'évolution. Notre objectif dans le troisième chapitre est de proposer une méthode pour lui éviter la description de la courbe de reparamétrisation. La reparamétrisation permet de contrôler la cinétique du mouvement[1]; en d'autres termes la vitesse d'évolution des paramètres le long de la courbe d'interpolation.

Le chapitre quatre sera l'occasion de regrouper les résultats précédents. Nous montrerons comment contrôler interactivement une animation au travers d'une seule fenêtre de dialogue, sans passer par une multitude de fenêtres. L'adaptation de ces techniques à la modélisation interactive des cylindres généralisés concluera cette première partie.

[1]le terme de dynamique du mouvement est parfois employé, particulièment dans les textes anglo-saxons (*motion dynamics*); nous l'éviterons afin d'éviter l'ambiguïté avec les animations basées sur la dynamique des solides

Chapitre 1

Le contrôle des orientations

1.1 Introduction

Le problème de l'interpolation est bien résolu en ce qui concerne les positions, essentiellement par l'emploi de courbes splines. En ce qui concerne les orientations, de nombreuses avancées ont été réalisées ces dernières années, en particulier, grâce à l'introduction des quaternions. Cependant, comme on le verra plus loin, le contrôle de l'interpolation est encore très limité.

Ce chapitre reprend les principes exposés dans [Han92b] et [Han92a] . Il s'organise de la façon suivante :

section 1.1 Les différentes représentations des orientations sont expliquées, mais cette section est surtout consacrée à la présentation des quaternions.

section 1.2 Diverses méthodes permettant d'interpoler linéairement entre deux orientations sont exposées. En particulier, nous explicitons une formule d'interpolation, ce qui à notre connaissance, n'a jamais été fait.

section 1.3 Cette partie décrit les solutions proposées par différents auteurs pour interpoler des orientations par des courbes splines.

section 1.4 Nous proposons une méthode d'interpolation basée sur les logarithmes et exponentielles de quaternions.

section 1.5 Nous détaillons une technique permettant d'éditer en temps réel des courbes interpolant des orientations. Un parallèle avec les méthodes utilisées pour les positions nous permet d'introduire la notion de tangente sphérique.

section 1.6 C'est une application de la section précédente. Nous y montrons comment interpoler entre différentes positions-clés d'un bras articulé.

1.1.1 Représentations des orientations

De même qu'une position est déterminée par une translation à partir d'un référentiel donné, une orientation est définie comme une rotation par rapport à une orientation de référence. Contrairement au cas des positions qui sont unanimement représentées par des vecteurs, il existe de nombreux moyens permettant de représenter une orientation (ou une rotation). Le choix de ces représentations dépend souvent de l'usage que l'on veut faire des rotations. Parmi les représentations possibles d'une rotation (ou d'une orientation), nous pouvons citer :

- Un axe de rotation et un angle. L'axe est par définition invariant.

- Une matrice de rotation (matrice 3×3 ou 4×4 en coordonnées homogènes). Si les matrices de transformation sont très répandues dans les parties modélisation et rendu de la synthèse d'images, elles sont peu adaptées à l'animation. En particulier, l'interpolation de deux matrices de rotation fournit rarement une matrice orthogonale.

- Les angles d'Euler. Dans ce cas, une rotation quelconque est définie par la composition successive de trois rotations. La propriété caractéristique des angles d'Euler est que chacune des rotations est effectuée par rapport à un des axes de base du repère initial, la position de cet axe étant le résultat des rotations précédentes. Le choix des axes utilisés est fait par convention. On peut par exemple effectuer une première rotation par rapport à l'axe z, puis par rapport à l'axe x et à nouveau par rapport à l'axe z, ce dernier étant en général différent de l'axe z original. Bien qu'intensivement utilisés en animation, les angles d'Euler souffrent d'un phénomène de blocage inhérent à leur définition. Cet inconvénient majeur, connu en anglais sous le terme de *gimbal lock*, est expliqué plus en détail dans la deuxième partie de ce rapport, au paragraphe 5.3.1.

- Les quaternions. Le paragraphe qui suit leur est consacré.

- Les exponentielles de matrices antisymétriques. Nous en parlerons plus en détail dans le paragraphe 1.3.4.

Les conversions d'une représentation à une autre sont en général relativement simples. On peut les trouver dans [HM88] et [Sho85].

Contrairement aux représentations communément utilisées en synthèse d'images, l'interpolation de deux quaternions fournit le plus court chemin entre ces deux orientations. C'est là leur avantage essentiel pour l'animation. Dans le paragraphe qui vient, nous donnons la définition des quaternions ainsi que les différentes opérations qui s'y rapportent. Le lecteur pourra trouver de plus amples détails concernant les diverses formules en annexe A.

1.1.2 Qu'est-ce qu'un quaternion?

Les quaternions, introduits par Hamilton [Ham44], sont des éléments de $\mathcal{R}^4$. En fait les quaternions unitaires suffisent à représenter des rotations. Ils permettent aussi de représenter une orientation par rapport à une orientation de référence, de même qu'une translation définit une position relativement à un point de référence. Leur intérêt vient de ce qu'ils offrent une représentation des rotations indépendantes d'un quelconque repère, contrairement aux angles d'Euler qui dépendent des axes de rotation choisis.

Les quaternions sont des éléments de $\mathcal{R}^4$. Ils forment un groupe multiplicatif. Ainsi, à la somme des vecteurs de $\mathcal{R}^3$, correspond le produit des quaternions. De même, multiplier un vecteur par un scalaire α, revient, dans $\mathcal{S}^3$, à élever un quaternion à la puissance α.

Un quaternion est composé d'une partie réelle et d'une partie imaginaire. On le note:

$$q = [w, v]$$

où w, la partie réelle de q est un scalaire et v, la partie imaginaire, est un vecteur de $\mathcal{R}^3$.

Le produit de deux quaternions s'écrit:

$$q_1 q_2 = [w_1, v_1][w_2, v_2] = [w_1 w_2 - v_1 \cdot v_2, w_1 v_2 + w_2 v_1 + v_1 \wedge v_2] \tag{1.1}$$

où $\cdot$ est le produit scalaire et $\wedge$ le produit vectoriel.

De même que deux rotations ne commutent pas en général, le produit de deux quaternions n'est pas commutatif. En fait il l'est seulement quand les deux quaternions représentent des rotations autour d'un même axe. Notons aussi que du point de vue informatique, la composition des rotations est une opération plus rentable en utilisant le produit des quaternions que le produit des matrices (16 multiplications et 12 additions au lieu de 27 multiplications et 18 additions). Une technique de calcul plus complexe permet même de réaliser ce produit en 7 multiplications [Laf75]. Cependant le nombre d'additions nécessaires (une bonne cinquantaine) en limite l'intérêt, du moins au niveau temps de calcul.

L'inverse du quaternion q, appelé aussi quaternion conjugué, est le quaternion

$$q^{-1} = [w, -v]$$

L'inverse vérifie la propriété suivante:

$$(q_1 q_2)^{-1} = q_2^{-1} q_1^{-1}$$

Si les coordonnées de v sont x, y et z, la norme du quaternion q est donnée par:

$$\|q\|^2 = w^2 + x^2 + y^2 + z^2$$

Les quaternions unitaires suffisent à représenter des rotations. Ils sont définis sur la sphère unitaire de $\mathcal{R}^4$ appelée $\mathcal{S}^3$. Pour représenter une rotation d'axe unitaire $v = (x, y, z)$ et d'angle θ, on utilise le quaternion (cf annexe A.1.1):

$$q = \left[\cos\frac{\theta}{2}, \sin\frac{\theta}{2} v\right] \tag{1.2}$$

L'intérêt fondamental du produit des quaternions est qu'il correspond à la composition des rotations.

Remarque

Tout comme pour les produits de matrices, l'ordre dans lequel sont faits les produits de quaternions à une signification particulière. Si l'on veut appliquer n rotations successives $q_1, q_2, \ldots, q_{n-1}, q_n$, on utilisera le quaternion composé $q_n q_{n-1} \ldots q_2 q_1$. Les rotations sont alors effectuées autour d'axes mobiles (l'axe de rotation du quaternion q_i est relatif aux rotations déjà réalisées). Au contraire, le quaternion composé $q_1 q_2 \ldots q_{n-1} q_n$ permet d'effectuer ces rotations autour d'axes fixes (relatifs au repère initial). Le lecteur pourra se référer à [Rip91] pour une démonstration de ces propriétés. Quand on veut appliquer une rotation constituée de trois rotations successives, il faut en tenir compte. Une rotation basée sur les angles d'Euler est composée de trois rotations autour d'axes mobiles alors que d'autres formulations – par exemple les angles de roulis, tangage et lacet couramment employés en aéronautique – utilisent des axes fixes.

L'élévation à la puissance n d'un quaternion tel que celui défini en (1.2), s'écrit:

$$q^n = \left[\cos\frac{\theta n}{2}, \sin\frac{\theta n}{2} v\right]$$

Par exemple, si q exprime une rotation d'angle θ, $q^{1/2}$ est le quaternion représentant une rotation d'angle $\frac{\theta}{2}$ autour du même axe (cf annexe A.1.2).

Enfin, appliquer une rotation à un vecteur v consiste à considérer ce vecteur comme un quaternion de partie réelle nulle et à lui appliquer la transformation suivante (cf annexe A.1.1):

$$q[0, v]q^{-1} \tag{1.3}$$

où q est le quaternion représentant la rotation.

L'ensemble des opérations propres au groupe des quaternions que nous venons d'énumérer, ainsi que d'autres que nous introduirons plus loin, sont détaillées dans l'annexe A.

1.1.3 Logarithmes et exponentielles de quaternions

Dans ce paragraphe, nous introduisons les notions de logarithmes et d'exponentielles de quaternions. Elles seront à la base des méthodes d'interpolation présentées ultérieurement.

On définit l'exponentielle d'un quaternion par la formule habituelle dont la preuve de convergence est donnée dans [Bor87]:

$$\exp(q) = 1 + \frac{q}{1!} + \frac{q^2}{2!} + \ldots + \frac{q^n}{n!} + \ldots \tag{1.4}$$

Considérons le cas particulier où la partie réelle du quaternion est nulle, c'est à dire $Q = [0, V]$. On note ces quantités en majuscule pour indiquer qu'elles ne sont pas forcément

unitaires. En utilisant le produit des quaternions défini en (1.1), nous avons:

$$\begin{aligned} Q^{4n} &= [\|V\|^{4n}, 0] \\ Q^{4n+1} &= [0, \|V\|^{4n}V] = \left[0, \frac{V}{\|V\|}\|V\|^{4n+1}\right] \\ Q^{4n+2} &= [-\|V\|^{4n+2}, 0] \\ Q^{4n+3} &= [0, -\|V\|^{4n+2}V] = \left[0, -\frac{V}{\|V\|}\|V\|^{4n+3}\right] \end{aligned}$$

En remplaçant les puissances successives de q dans (1.4):

$$\exp(Q) = \left[1 - \frac{\|V\|^2}{2!} + \frac{\|V\|^4}{4!} - \ldots, \frac{V}{\|V\|}\left(\|V\| - \frac{\|V\|^3}{3!} + \ldots\right)\right]$$

Et finalement:

$$\exp(Q) = \left[\cos\|V\|, \frac{V}{\|V\|}\sin\|V\|\right] \tag{1.5}$$

L'exponentielle d'un quaternion de partie réelle nulle est donc un quaternion unitaire. Si q désigne à nouveau un quaternion unitaire, on note $exp(Q) = exp([0, V]) = q$. Réciproquement, le logarithme d'un quaternion unitaire q est un quaternion de partie réelle nulle ([0,V]). Dans la suite, on ne conserve que la partie imaginaire du logarithme. C'est un vecteur de dimension trois. En effet, le logarithme V d'un quaternion unitaire $q = [w, v]$ est donné par:

$$\begin{aligned} \|V\| &= \cos^{-1} w \\ V &= \frac{\|V\|}{\sin\|V\|}v \end{aligned}$$

c'est à dire (cf annexe A.1.4):

$$\log q = \begin{cases} \frac{\cos^{-1} w}{\sqrt{1-w^2}}v & \text{si} \quad w \neq 1 \\ 0 & \text{sinon} \end{cases}$$

En comparant les expressions (1.2) et (1.5), une rotation d'angle θ et d'axe v, définie par le quaternion $q = [\cos\frac{\theta}{2}, \sin\frac{\theta}{2}v]$ peut aussi être représentée par le vecteur qui est le logarithme de ce quaternion:

$$\log q = \frac{\theta}{2}v$$

ce qui nous permet par ailleurs d'écrire q de manière condensée sous la forme[1]:

$$q = e^{\frac{\theta}{2}v}$$

Il est important de noter que l'addition des logarithmes de quaternions n'est, en général, pas commutative, i.e. $e^{\log q_1 + \log q_2}$ est en général différent de $e^{\log q_1}e^{\log q_2} = q_1 q_2$. C'est une conséquence directe de la non-commutativité du produit des quaternions. En fait, la commutativité n'est vérifiée que si les deux logarithmes sont colinéaires.

[1] Il est sous-entendu dans ce qui suit que l'on appelle exponentielle d'un quaternion, l'exponentielle d'un quaternion de partie réelle nulle. On la notera indifféremment $\exp(q)$ ou e^q

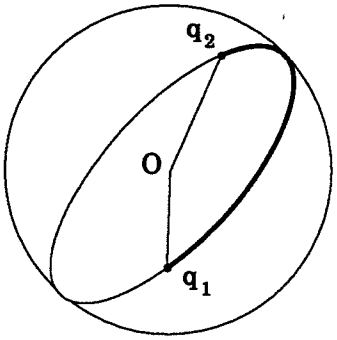


FIG. 1.1 L'interpolation linéaire entre deux quaternions décrit un arc de cercle sur $\mathcal{S}^3$.

1.2 Interpolation linéaire entre deux orientations

1.2.1 Interpolation linéaire des positions

Nous désirons définir la trajectoire linéaire permettant de passer d'une position p_1 à une position p_2. Le vecteur qui transforme p_1 en p_2 est $p_2 - p_1$. L'interpolation consiste à translater progressivement p_1 par ce vecteur, ce qui nous fournit le chemin :

$$p(u) = p_1 + u(p_2 - p_1)$$

où u varie de 0 à 1. Par la suite, nous noterons cette interpolation $lerp(p_1, p_2, u)$, pour *linear interpolation*. De même, $mid(p_1, p_2) = lerp(p_1, p_2, \frac{1}{2})$ désigne le milieu du segment (p_1, p_2).

1.2.2 Interpolation linéaire des orientations

Les quaternions unitaires étant définis sur la sphère unitaire de $\mathcal{R}^4$, l'interpolation linéaire entre deux quaternions décrit un arc de cercle sur $\mathcal{S}^3$ (voir figure 1.1). C'est ce que Shoemake [Sho85] appelle une interpolation linéaire sphérique ou *slerp* pour *spherical linear interpolation*. Nous reprendrons ici sa notation, $slerp(q_1, q_2, u)$ désignant l'interpolation entre q_1 et q_2, u variant entre 0 et 1. De même, $smid(q_1, q_2) = slerp(q_1, q_2, \frac{1}{2})$ désigne le milieu "sphérique" de deux quaternions.

Plusieurs formulations ont été proposées pour décrire cette interpolation sur la sphère unitaire de $\mathcal{R}^4$.

La première utilise les propriétés du groupe des quaternions. En effectuant un parallèle avec l'interpolation des positions dans $\mathcal{R}^3$, nous allons obtenir une première méthode pour interpoler sur $\mathcal{S}^3$. La rotation qui permet de passer de l'orientation q_1 à l'orientation q_2 est définie par le quaternion $q_1^{-1}q_2$. Une fraction de cette rotation est fournie par $(q_1^{-1}q_2)^u$. Enfin, en appliquant celle-ci au quaternion q_1, on obtient la formule (dont la construction est donnée en annexe A.1.3) :

$$slerp(q_1, q_2, u) = q_1 \left(q_1^{-1}q_2\right)^u$$

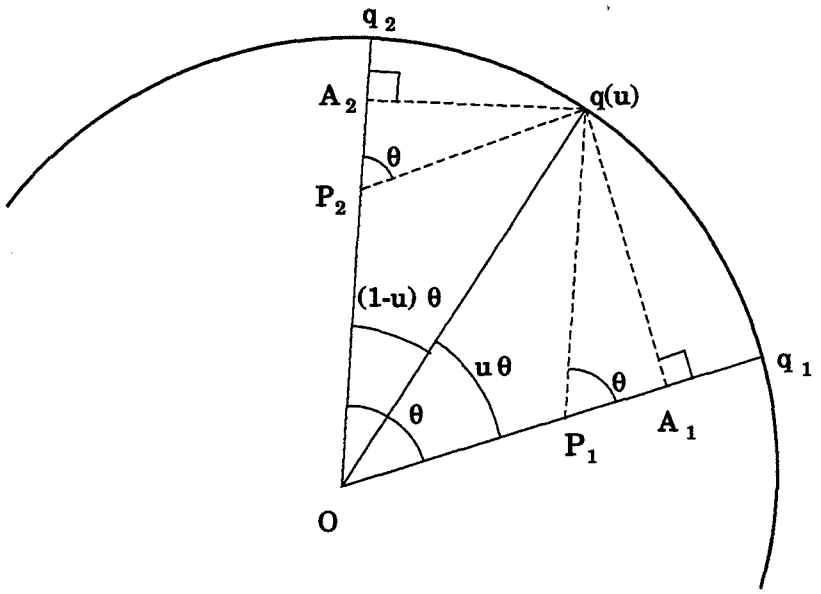


FIG. 1.2 Plan de l'interpolation linéaire

Cette formule est informatiquement peu facile d'emploi aussi préfère-t-on généralement utiliser cette autre formulation, fondée sur la géométrie dans $\mathcal{R}^4$:

$$slerp(q_1, q_2, u) = \frac{\sin(1-u)\theta}{\sin\theta} q_1 + \frac{\sin u\theta}{\sin\theta} q_2 \tag{1.6}$$

où $0 \leq u \leq 1$ et $\cos\theta = q_1 \cdot q_2$, $\cdot$ désignant le produit scalaire des quaternions ($q_1 \cdot q_2 = w_1 w_2 + v_1 \cdot v_2$).

Un article de Heise et MacDonald [HM88] démontre la validité de (1.6). Ils prouvent que le quaternion interpolé se situe bien sur $\mathcal{S}^3$ et que l'angle parcouru à l'instant u est bien $u\theta$. Cependant, l'origine de cette formule, introduite par Shoemake dans [Sho85], n'a à notre connaissance, jamais été explicitée.

Pour cela considérons la figure (1.2) représentant l'hyperplan (O, q_1, q_2) de la figure (1.1). Dans la démonstration qui suit, les quaternions sont traités comme des vecteurs unitaires de $\mathcal{R}^4$.

Pour un u donné, l'angle de l'arc entre q_1 et $q(u)$ est $u\theta$. Notons O le centre de $\mathcal{S}^3$, I la position de $q(u) = slerp(q_1, q_2, u)$, P_1 la projection de I sur Oq_1 parallèlement à Oq_2.

Concernant l'arc de centre O reliant q_1 à $q(u)$ et d'angle $u\theta$, nous avons :

$$\sin u\theta = \frac{\overline{A_1 I}}{\overline{OI}}$$

$$\sin\theta = \frac{\overline{A_1 I}}{\overline{P_1 I}}$$

En considérant les deux équations précédentes et en remarquant que $q(u)$ est un quaternion unitaire, c'est à dire que $\overline{OI} = 1$, nous avons :

$$\overline{P_1 I} = \overline{OI}\frac{\sin u\theta}{\sin\theta} = \frac{\sin u\theta}{\sin\theta} \tag{1.7}$$

Concernant l'arc de centre O reliant I à q_2 et d'angle $(1-u)\theta$, nous avons

$$\sin(1-u)\theta = \frac{\overline{A_2 I}}{\overline{OI}}$$

$$\sin\theta = \frac{\overline{A_2 I}}{\overline{P_2 I}}$$

d'où

$$\overline{P_2 I} = \frac{\sin(1-u)\theta}{\sin\theta} \tag{1.8}$$

Pour achever la démonstration, il suffit de remarquer que

$$q(u) = \overline{OP_1}q_1 + \overline{P_1 I}q_2 = \overline{P_2 I}q_1 + \overline{P_1 I}q_2$$

et d'utiliser (1.7) et (1.8) pour retrouver l'équation (1.6).

1.2.3 Utilisation des logarithmes de quaternions

Cette partie introduit l'idée fondamentale qui permettra par la suite de créer des courbes splines sur l'espace des orientations. Pour l'instant, on se limite à l'interpolation linéaire de deux orientations.

Quand les deux quaternions à interpoler représentent des rotations d'axes concourants (quand les rotations commutent), on peut envisager d'interpoler les logarithmes de quaternions, puis de retrouver le quaternion interpolé par l'exponentielle. L'interpolation s'écrit alors :

$$slerp(q1, q2, u) = \exp(lerp(\log q_1, \log q_2, u))$$

Evidemment, si les axes ne sont pas colinéaires, cette solution n'est plus exacte. Cependant, on peut constater que l'approximation ainsi obtenue n'est pas loin de l'interpolation linéaire sphérique exacte. On peut s'en rendre compte sur la figure (1.3). On y a représenté l'image d'un vecteur par une suite de quaternions ainsi que les interpolations linéaires entre deux orientations successives de ce vecteur. Cela nous permet de visualiser un phénomène propre à la dimension quatre. Les arcs de cercle représentés par des croix sont obtenus avec la formule (1.6) et représentent l'interpolation linéaire exacte. L'interpolation approchée, utilisant les logarithmes de quaternions, est affichée avec des petits cercles.

Evidemment, il serait souhaitable d'établir une preuve théorique donnant une estimation de l'erreur entre l'interpolation exacte et l'interpolation approchée, ou à défaut d'effectuer des statistiques sur cette erreur. Le faible écart rencontré dans la pratique entre la solution exacte et celle utilisant les logarithmes de quaternions permet d'envisager l'utilisation des exponentielles et logarithmes de quaternions pour la création de courbes splines

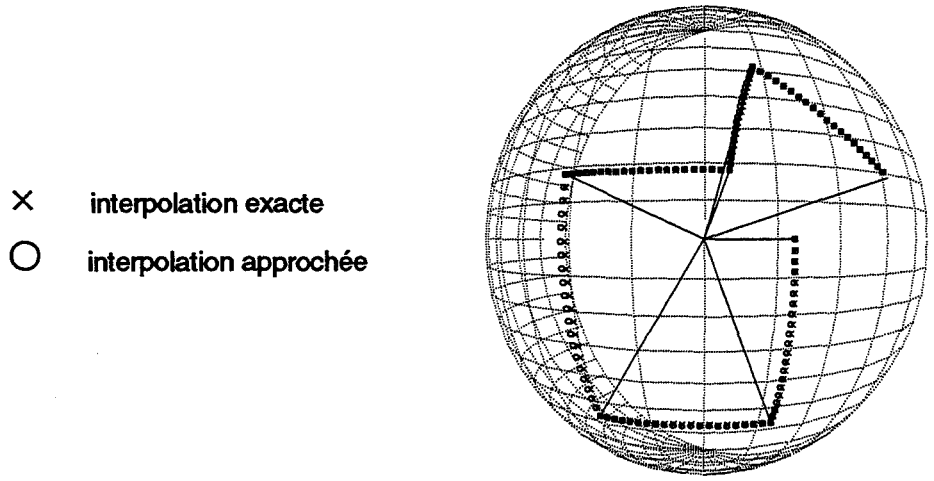


FIG. 1.3 Interpolation linéaire exacte et approchée

dans $\mathcal{S}^3$. Il suffit pour cela de remplacer dans la formule précédente l'interpolation linéaire des logarithmes de quaternion par tout autre technique d'interpolation tri-dimensionnelle connue.

C'est ce que nous exposerons dans la section 1.4, après avoir revu l'ensemble des méthodes existantes à ce jour dans la section ci-dessous.

1.3 Interpolation des orientations

Dans le paragraphe précédent, nous avons traité le cas de l'interpolation entre deux, et seulement deux, orientations-clés. En animation, nous sommes confrontés en permanence à des séquences pouvant contenir plusieurs dizaines d'orientations-clés, ce qui requiert la spécification de techniques de lissage. Ces dernières font appel aux splines d'interpolation, par opposition aux splines d'approximation, plus fréquemment utilisées en modélisation.

Avant l'introduction des quaternions, deux techniques étaient principalement utilisées pour interpoler des orientations.

La première, utilisant les angles d'Euler, consiste à interpoler les valeurs de ces angles. L'inconvénient avec les angles d'Euler est que les trois rotations ne sont pas indépendantes les unes des autres. Or, interpoler les angles d'Euler revient à interpoler un vecteur de dimension trois, en traitant chacune de ses composantes de façon indépendante. Le résultat est un chemin sinueux entre les deux orientations alors que le chemin idéal est un arc de cercle reliant les deux quaternions (figure 1.4).

La deuxième solution est d'interpoler les éléments des différentes matrices spécifiant chacune des orientations-clé. Evidemment, les matrices interpolées ne sont en général plus orthogonales, ce qui nécessite, dans un post-traitement, d'orthogonaliser ces matrices. Des singularités ou discontinuités peuvent ici aussi apparaître lors de ce processus [Duf86].

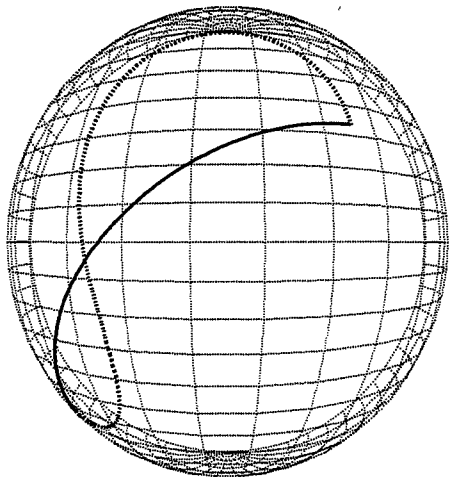


FIG. 1.4 Comparaison de l'interpolation linéaire entre des quaternions (en trait plein) et entre des angles d'Euler (en pointillés). La première orientation est définie par les angles en degrés x-z-x $(100, 140, 80)$, la deuxième par les angles $(50, -120, 180)$.

En ce qui concerne l'utilisation des quaternions, les auteurs que nous citerons par la suite se sont restreints aux courbes splines qui ont l'avantage d'être locales, c'est à dire que la modification d'un point de contrôle n'affecte qu'un nombre limité de morceaux de courbes dans le voisinage du point considéré. C'est particulièrement utile en animation. On peut changer une petite partie de la trajectoire d'un objet sans perturber les parties éloignées de la trajectoire globale.

La plupart des splines à caractère local peuvent être définies par un système d'équations linéaires précisant la relation entre les points de contrôle et leurs coefficients polynomiaux. Ces équations peuvent être représentées sous forme matricielle.

En fait, on décompose généralement la trajectoire globale en un ensemble de splines d'ordre trois, ou splines cubiques. Ce type de courbe est construit à partir de quatre points de contrôle avec des polynômes d'ordre trois. Les quadruplets définissant ces splines cubiques sont obtenus à partir de l'ensemble des points de contrôle initiaux.

L'expression générale d'une spline cubique peut s'écrire sous forme matricielle de la façon suivante :

$$P_i(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} M_{spline} \begin{pmatrix} P_i \\ P_{i+1} \\ P_{i+2} \\ P_{i+3} \end{pmatrix}$$

La matrice M_{spline} contient les coefficients des quatre polynômes d'interpolation. Elle dépend du type de courbe choisi. Ces splines sont uniformes, chaque segment de courbe étant défini sur un intervalle de longueur 1 (le paramètre u varie entre 0 et 1).

De nombreuses familles de splines existent. Nous nous contentons d'indiquer les ma-

trices des splines cubiques dont nous nous servirons par la suite :

Spline cardinale : La courbe passe par les points P_{i+1} et P_{i+2}, les points P_i et P_{i+3} déterminant la demi-tangente à droite en P_{i+1} et la demi-tangente à gauche en P_{i+2}. La tangente à droite en P_{i+1} est $D_{i+1} = \alpha(P_{i+2} - P_i)$. La tangente à gauche en P_{i+2} est $G_{i+2} = \alpha(P_{i+1} - P_{i+3})$. Le paramètre α permet d'influer sur le module des tangentes, ce qui se traduit géométriquement par une variation de la tension de la courbe. Quand $\alpha = \frac{1}{2}$, on parle de splines de Catmull-Rom.

$$M_{c-spline} = \begin{pmatrix} -\alpha & 2-\alpha & \alpha-2 & \alpha \\ 2\alpha & \alpha-3 & 3-2\alpha & -\alpha \\ -\alpha & 0 & \alpha & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{1.9}$$

Bspline : Cette courbe ne passe par aucun des points de contrôle mais passe dans le voisinage de chacun de ces points. Elle est incluse dans l'enveloppe convexe des points de contrôle.

$$M_{Bspline} = \frac{1}{6}\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{pmatrix} \tag{1.10}$$

Courbe de Bézier : Elle passe par les deux points extrêmes P_i et P_{i+3}, les tangentes en ces points étant déduites des points de contrôle P_{i+1} et P_{i+2}. Une courbe de Bézier est aussi incluse dans l'enveloppe convexe de ses points de contrôle.

$$M_{Bezier} = \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \tag{1.11}$$

Interpolation d'Hermite : L'interpolation d'Hermite se distingue des autres courbes. Ici les paramètres sont deux points de contrôle P_i et P_{i+1} ainsi que deux vecteurs représentant la tangente à droite D_i en P_i et la tangente à gauche G_{i+1} en P_{i+1} (des détails sont donnés sur l'interpolation d'Hermite dans l'annexe B).

$$M_{Hermite} = \begin{pmatrix} 2 & 1 & -2 & -1 \\ -3 & -2 & 3 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \tag{1.12}$$

1.3.1 Courbes de Bézier

Nous allons commencer par évoquer la méthode proposée par Shoemake [Sho85] pour interpoler des orientations. Son choix s'est porté sur les courbes de Bézier qui, comme

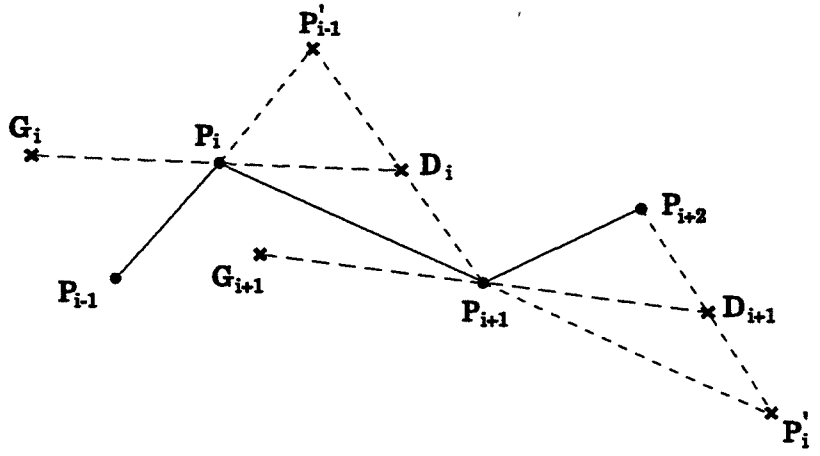


FIG. 1.5 Choix des tangentes pour l'interpolation de Bézier

on le verra plus loin, ont l'énorme avantage de pouvoir être construites par des méthodes géométriques.

Tout d'abord, voyons dans $\mathcal{R}^3$, comment il décompose la courbe interpolant les $n+2$ points de contrôle initiaux en courbes de Bézier d'ordre trois. Comme on le sait, les courbes de Bézier ne passent que par les points de contrôle extrémaux. De façon à obtenir une courbe passant par tous les points de contrôle, il suffit de considérer que chaque morceau de courbe dépend d'un quadruplet $(P_i, D_i, G_{i+1}, P_{i+1})$. Les points D_i et G_{i+1} représentent respectivement la tangente à droite en P_i et la tangente à gauche en P_{i+1} (voir figure 1.5).

La construction géométrique de la tangente à droite D_i en P_i se fait comme suit. On commence par déterminer le symétrique P'_{i-1} de P_{i-1} par rapport à P_i. Le milieu du segment (P'_{i-1}, P_{i+1}) fournit le résultat cherché. Le vecteur G_i est le symétrique de D_i par rapport à P_i, ce qui assure la continuité à l'ordre un entre les différents morceaux de courbe.

Ces constructions peuvent être effectuées de la même façon pour les quaternions, en remplaçant la notion de segment par celle d'arc de cercle sur $\mathcal{S}^3$.

Intéressons-nous maintenant à la construction géométrique d'une courbe de Bézier. Une courbe de Bézier de degré d est définie par:

$$P_i(u) = \sum_{i=0}^{n} P_i B_{i,d}(u)$$

avec

$$B_{i,d}(u) = \begin{pmatrix} d \\ i \end{pmatrix} u^i (1-u)^{d-i}$$

Une propriété importante des courbes de Bézier est la possiblité de les définir de manière récursive:

$$P_i(u) = \sum_{i=0}^{d-1} B_{i,d-1}(u) lerp(P_i, P_{i+1}, u)$$

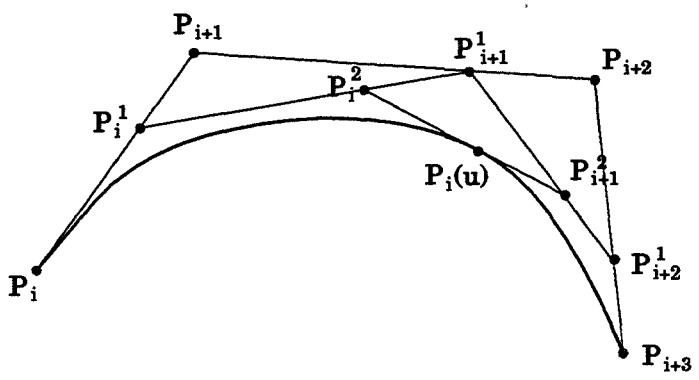


FIG. 1.6 Construction géométrique d'une courbe de Bézier d'ordre trois.

$$= \sum_{i=0}^{d-2} B_{i,d-2}(u) lerp(lerp(P_i, P_{i+1}, u), lerp(P_{i+1}, P_{i+2}, u), u)$$

etc...

L'intérêt de ce qui précède est que l'on peut construire géométriquement une courbe de Bézier uniquement à partir d'interpolations linéaires. Cette méthode a été proposée par De Casteljau. A l'ordre trois, l'évaluation d'un point de la courbe pour un u donné requiert exactement l'évaluation de six interpolations linéaires, en trois étapes distinctes (figure 1.6). Les chiffres placés en exposant à chaque point indiquent le numéro de l'étape considérée:

étape 1 :

$$\begin{aligned} P_i^1 &= lerp(P_i, P_{i+1}, u) \\ P_{i+1}^1 &= lerp(P_{i+1}, P_{i+2}, u) \\ P_{i+2}^1 &= lerp(P_{i+2}, P_{i+3}, u) \end{aligned}$$

étape 2 :

$$\begin{aligned} P_i^2 &= lerp(P_i^1, P_{i+1}^1, u) \\ P_{i+1}^2 &= lerp(P_{i+1}^1, P_{i+2}^1, u) \end{aligned}$$

étape 3 :

$$P_i^3 = P_i(u) = lerp(P_i^2, P_{i+1}^2, u)$$

C'est à partir de cette technique de construction géométrique que Shoemake construit une courbe de Bézier sur $\mathcal{S}^3$. Simplement, il remplace l'interpolation linéaire de $\mathcal{R}^3$ par l'interpolation linéaire sphérique introduite au §1.2.

1.3.2 B-splines

Cette contribution, due à Duff [Duf86], utilise une méthode de décomposition des courbes B-splines (équation 1.10). Comme dans le paragraphe précédent, nous ne nous intéressons pour commencer qu'à des courbes dans $\mathcal{R}^3$.

Son objectif est de décomposer une courbe B-spline $S(t)$ définie par quatre points de contrôle S_i, S_{i+1}, S_{i+2} et S_{i+3} en deux courbes $G(u)$ et $D(u)$ définies par les nouveaux quadruplets G_i, G_{i+1}, G_{i+2}, G_{i+3} et D_i, D_{i+1}, D_{i+2}, D_{i+3}, où $u = 2t$ (figure 1.7). Les points de contrôle '•' désignent les points initiaux, ceux notés '×' désignent les points G_i et les 'o' représentent les D_i.

$S(t)$ étant une spline cubique, $S(t) = At^3 + Bt^2 + Ct + D$, et on a :

$$\begin{aligned} G(u) &= S(\frac{t}{2}) = A\left(\frac{t}{2}\right)^3 + B\left(\frac{t}{2}\right)^2 + C\left(\frac{t}{2}\right) + D \\ &= \begin{pmatrix} t^3 & t^2 & t & 1 \end{pmatrix} M_G \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} \end{aligned}$$

avec

$$M_G = \frac{1}{8}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 8 \end{pmatrix}$$

et

$$\begin{aligned} D(u) &= S(\frac{t+1}{2}) = A\left(\frac{t+1}{2}\right)^3 + B\left(\frac{t+1}{2}\right)^2 + C\left(\frac{t+1}{2}\right) + D \\ &= \begin{pmatrix} t^3 & t^2 & t & 1 \end{pmatrix} M_D \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} \end{aligned}$$

avec

$$M_D = \frac{1}{8}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 2 & 0 & 0 \\ 3 & 4 & 4 & 0 \\ 1 & 2 & 4 & 8 \end{pmatrix}$$

On peut alors calculer les nouveaux points de contrôle des courbes $G(u)$ et $D(u)$ à partir de ceux de S en multipliant ces derniers par :

$$M_{Bspline_G} = M_{Bspline}^{-1} M_G M_{Bspline} = \frac{1}{8}\begin{pmatrix} 4 & 4 & 0 & 0 \\ 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 1 & 6 & 1 \end{pmatrix}$$

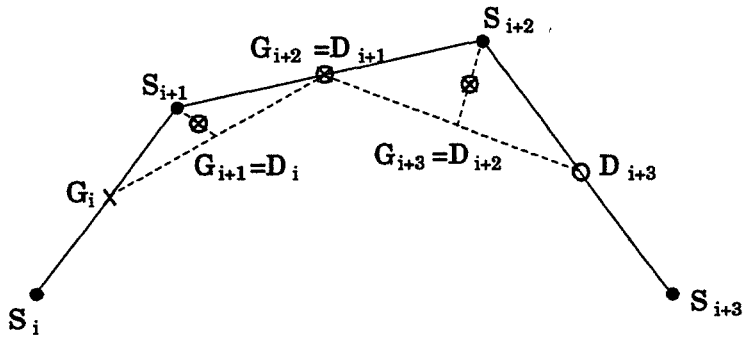


FIG. 1.7 Subdivision géométrique d'une B-spline.

et

$$M_{Bspline_D} = M_{Bspline}^{-1} M_D M_{Bspline} = \frac{1}{8}\begin{pmatrix} 1 & 6 & 1 & 0 \\ 0 & 4 & 4 & 0 \\ 0 & 1 & 6 & 1 \\ 0 & 0 & 4 & 4 \end{pmatrix}$$

A partir des quatre points de contrôle de la courbe initiale, on obtient ainsi deux couples de quatre points définissant deux nouvelles B-splines. On peut remarquer que les trois dernières lignes de $M_{Bspline_G}$ sont identiques aux trois premières de $M_{Bspline_D}$ ce qui indique que les deux nouvelles courbes ont trois points de contrôle en commun (figure 1.7) :

$$\begin{aligned} G_i &= \frac{S_i + S_{i+1}}{2} \\ G_{i+1} = D_i &= \frac{S_i + 6S_{i+1} + S_{i+2}}{8} \qquad (1.13) \\ G_{i+2} = D_{i+1} &= \frac{S_{i+1} + S_{i+2}}{2} \\ G_{i+3} = D_{i+2} &= \frac{S_{i+1} + 6S_{i+2} + S_{i+3}}{8} \qquad (1.14) \\ D_{i+3} &= \frac{S_{i+2} + S_{i+3}}{2} \end{aligned}$$

En remarquant que l'on peut aussi écrire (1.13) et (1.14) sous une forme ne faisant intervenir que les milieux entre deux points :

$$G_{i+1} = D_i = \frac{\frac{S_i+S_{i+1}}{2} + \frac{S_{i+1}+S_{i+2}}{2}}{2} + S_{i+1} = mid(mid(G_i, G_{i+2}), S_{i+1})$$

$$G_{i+3} = D_{i+2} = \frac{\frac{S_{i+1}+S_{i+2}}{2} + \frac{S_{i+2}+S_{i+3}}{2}}{2} + S_{i+2} = mid(mid(D_{i+1}, D_{i+3}), S_{i+2})$$

on obtient ainsi une méthode de décomposition d'une courbe B-spline ne faisant intervenir que des opérations que l'on sait réaliser sur $\mathcal{S}^3$.

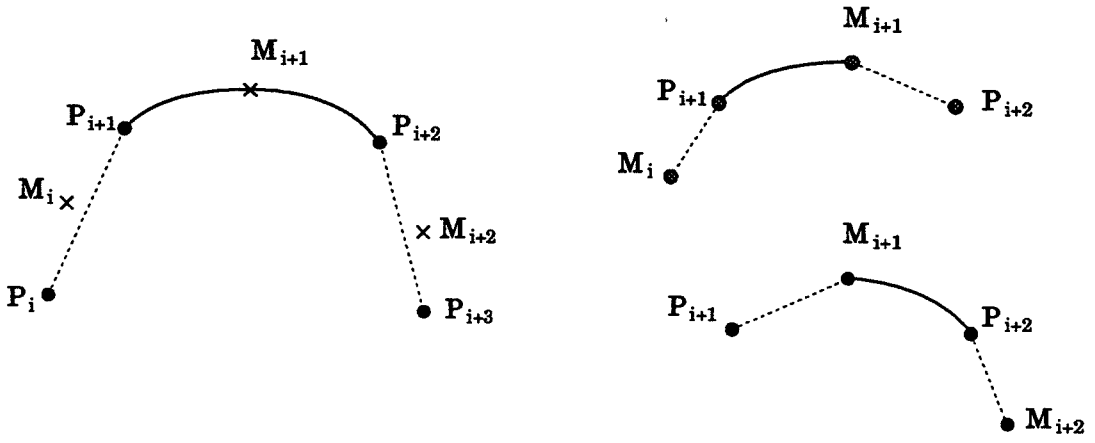


FIG. 1.8 Subdivision géométrique d'une spline cardinale

Là encore, l'application aux quaternions est immédiate, il suffit de remplacer *mid* par *smid*. La décomposition peut se poursuivre récursivement sur les deux nouvelles courbes ainsi obtenues.

1.3.3 Splines cardinales

Une autre approche, utilisant cette fois les splines cardinales ou les B-splines avec tension est proposée par Pletincks [Ple88] [Ple89]. Il se propose de construire sur la sphère de $\mathcal{R}^4$ deux familles de splines : les splines cardinales et les splines "avec tension". Nous nous bornerons ici à montrer la méthode concernant les splines cardinales. Le principe pour les B-splines avec tension est identique. Le principe fondamental est une fois encore de n'utiliser que des constructions géométriques telles que *mid* ou *lerp*.

La méthode consiste à subdiviser récursivement chaque spline en deux autres splines (figure 1.8). Rappelons que la spline cubique cardinale S_i de points de contrôle P_i, P_{i+1}, P_{i+2} et P_{i+3} passe par les points P_{i+1} et P_{i+2}. Pour effectuer la décomposition, on commence par construire les milieux $M_i = S_i(\frac{1}{2})$ de chacune des courbes S_i. Ensuite, on va scinder $S_i(u)$ en deux courbes définies par les deux quadruplets (M_i, P_{i+1}, M_{i+1}, P_{i+2}) et (P_{i+1}, M_{i+1}, P_{i+2}, M_{i+2}). La courbe initiale joignant P_{i+1} et P_{i+2} est ainsi décomposée en deux courbes joignant P_{i+1} à M_{i+1} et M_{i+1} à P_{i+2}.

L'opération est répétée récursivement avec chacun des deux nouveaux quadruplets jusqu'à obtenir une précision suffisante. La courbe finale est construite en reliant entre eux les points de contrôle des courbes du niveau le plus profond de la récursivité.

Dans le cas des splines cardinales, en utilisant la matrice (1.9), le milieu M_i de la spline est donné par (voir figure 1.9) :

$$\begin{aligned} M_i &= \frac{-\alpha}{8}P_i + \left(\frac{1}{2}+\frac{\alpha}{8}\right)P_{i+1} + \left(\frac{1}{2}+\frac{\alpha}{8}\right)P_{i+2} - \frac{\alpha}{8}P_{i+3} \\ &= lerp(mid(P_{i+1},P_{i+2}), mid(P_i,P_{i+3}), 1+\frac{\alpha}{4}) \end{aligned}$$

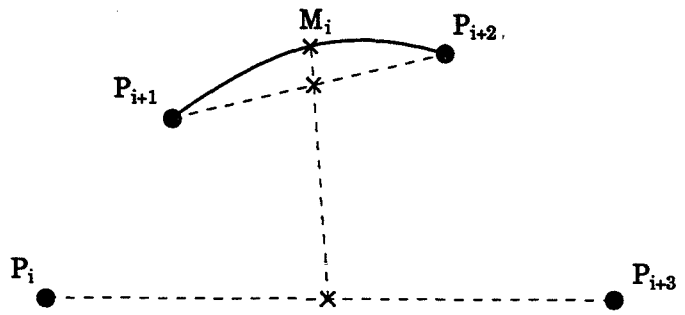


FIG. 1.9 Construction géométrique du milieu d'une spline cardinale

Les milieux successifs peuvent donc être construits géométriquement. L'application aux quaternions est immédiate. Il suffit de remplacer les opérations géométriques de $\mathcal{R}^3$ par les opérations équivalentes sur $\mathcal{S}^3$, en l'occurrence *smid* et *slerp* (cf §1.2).

Cependant, Pletincks souligne qu'au delà de la première subdivision, les milieux construits géométriquement ne restent pas forcément sur la spline elle-même. Le résultat n'est donc qu'une approximation de la spline cardinale désirée.

1.3.4 Matrices antisymétriques

Nous présentons dans cette partie l'approche présentée par Yahia et Gagalowicz [GY89, YG89] qui se distingue nettement des précédentes. Le cœur de leur méthode repose sur la paramétrisation des rotations par des exponentielles de matrices antisymétriques. Par certains aspect, cette approche peut être comparée à celle que nous présenterons au paragraphe 1.4.

On définit la matrice antisymétrique A^v associée au vecteur v de coordonnées v_1, v_2 et v_3 de la façon suivante :

$$A^v = \begin{pmatrix} 0 & v_3 & v_2 \\ -v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{pmatrix}$$

Yahia et Gagalowicz font remarquer qu'il y a une relation étroite entre l'exponentielle des matrices antisymétriques et les rotations représentées par des quaternions. En effet, comme le souligne [Bor87], une rotation d'angle θ et d'axe v peut-être aussi bien

représentée par le quaternion $q = e^{\frac{\theta}{2}v}$ que par la matrice $R = e^{\frac{\theta}{2}A^v}$[2]. Ainsi, ils peuvent paramétrer l'espace des quaternions unitaires (la sphère de $\mathcal{R}^4$) par un espace plat à trois dimensions : $\mathcal{R}^3$.

Ils exhibent la propriété importante suivante :

Si R est une rotation d'angle θ et d'axe de rotation v, alors le plus court chemin entre $Id_{\mathcal{R}^3}$[3] et R est donné par le chemin :

$$u \rightarrow e^{uA^v} \quad 0 \leq u \leq \theta$$

Ils en déduisent que le plus court chemin entre deux orientations d'axes v_1 et v_2, d'angles θ_1 et θ_2 est l'image par l'exponentielle des matrices antisymétriques du segment de droite reliant les vecteurs $\theta_1 v_1$ et $\theta_2 v_2$. D'après eux, la matrice de rotation interpolant entre ces deux orientations peut donc être représentée par le chemin suivant :

$$u \rightarrow e^{A^{lerp(v_1\theta_1, v_2\theta_2, u)}}$$

Malheureusement, ce n'est pas le cas, toujours en raison de la non commutativité des rotations. Cette paramétrisation est en fait équivalente à celle, présentée au paragraphe 1.2.3, utilisant les logarithmes de quaternions. Une rotation e^{A^v} peut aussi être représentée par e^v, ou la premiere exponentielle est celle des matrices et la deuxième celle des quaternions. On peut donc penser que l'erreur engendrée lors d'une interpolation est la même que celle observée sur la figure (1.3).

Pour lever toute incertitude, considérons une rotation R_1 d'axe $v_1 = (0,0,1)$ et d'angle $\theta_1 = \frac{\pi}{2}$ et une rotation R_2 d'axe $v_2 = (1,0,0)$ et d'angle $\theta_2 = \frac{\pi}{2}$ et montrons qu'à mi-chemin, l'orientation interpolée avec les exponentielles de matrices antisymétriques est différente de l'orientation interpolée avec (1.6).

R_1 (resp. R_2) est représentée par le quaternion $q_1 = \left[\frac{\sqrt{2}}{2}, (0,0,\frac{\sqrt{2}}{2})\right]$ (resp. $q_1 = \left[\frac{\sqrt{2}}{2}, (\frac{\sqrt{2}}{2},0,0)\right]$). En appliquant la formule d'interpolation linéaire sphérique des quaternions (1.6), à mi-chemin, on obtient le quaternion $q(\frac{1}{2}) = [0.816, (0.408, 0, 0.408)]$ que l'on peut convertir sous forme matricielle en

$$R_{\frac{1}{2}} = \begin{pmatrix} 0.67 & 0.67 & 0.33 \\ -0.67 & 0.33 & 0.67 \\ 0.33 & -0.67 & 0.67 \end{pmatrix} \tag{1.15}$$

[2] e désignant ici l'exponentielle des matrices antisymétriques définie de la façon suivante :

$$e^{A^v} = \sum_{i \geq 0} \frac{(A^v)^i}{i!}$$

Yahia et Gagalowicz montrent que l'évaluation de l'exponentielle d'une matrice antisymétrique peut en pratique être réalisée grâce à la formule suivante qui exprime la matrice résultat comme une somme finie, réduite à l'ordre deux :

$$e^{\theta A^v} = I + \frac{\sin\theta}{\theta} A^v + \frac{1-\cos\theta}{\theta^2} (A^v)^2$$

[3] $Id_{\mathcal{R}^3}$ désigne la matrice identité de $\mathcal{R}^3$

Appliquons maintenant la méthode énoncée plus haut sur les orientations représentées cette fois par les vecteurs de $\mathcal{R}^3$: $v_1 = (0, 0, \frac{\pi}{2})$ et $v_2 = (\frac{\pi}{2}, 0, 0)$.

L'interpolation linéaire entre v_1 et v_2 fournit à mi-chemin le vecteur :

$$v_{\frac{1}{2}} = v_1 + \frac{1}{2}(v_2 - v_1) = \left(\frac{\pi}{4}, 0, \frac{\pi}{4}\right)$$

Le calcul de l'exponentielle de la matrice antisymétrique associée à $v_{\frac{1}{2}}$ nous fournit le résultat suivant :

$$R_{\frac{1}{2}} = A^{v_{\frac{1}{2}}} = \begin{pmatrix} 0.72 & 0.63 & 0.28 \\ -0.63 & 0.44 & 0.63 \\ 0.28 & -0.63 & 0.72 \end{pmatrix} \tag{1.16}$$

Il suffit de comparer les matrices (1.15) et (1.16) pour se rendre compte que l'image de l'interpolation linéaire par les matrices antisymétriques ne passe pas par l'orientation exacte représentée par la matrice (1.15). On peut constater que ces deux matrices sont cependant presque semblables ; tout comme les logarithmes de quaternions, on obtient des trajectoires approximant bien les trajectoires exactes.

Revenons maintenant à l'interpolation des orientations par des courbes splines. Partant du principe que la paramétrisation de $\mathcal{R}^3$ par les exponentielles de matrices antisymétriques est valable, il suffit de construire dans $\mathcal{R}^3$ une courbe spline interpolant les $v_i\theta_i$, représentant les diverses orientations-clés. Ensuite le passage de $\mathcal{R}^3$ à l'espace des orientations est réalisé en calculant l'image par l'exponentielle des matrices antisymétriques de la trajectoire définie dans $\mathcal{R}^3$.

Ainsi, et c'est là l'avantage fondamental de cette méthode, il suffit de savoir éditer interactivement des courbes dans $\mathcal{R}^3$, pour modifier implicitement les trajectoires résultantes dans l'espace des orientations.

1.3.5 Minimisation de la courbure dans $\mathcal{S}^3$

Une contribution récente présentée dans [BCGH92] utilise un principe différent. Ils appliquent dans $\mathcal{S}^3$ une méthode de construction des splines dans un espace courbe quelconque [GK85]. Les auteurs construisent directement les splines dans l'espace non-euclidien des quaternions. Par analogie avec la définition des splines naturelles, le principe est de générer une trajectoire dans $\mathcal{S}^3$ qui minimise la courbure. En fait, ils approximent la courbure par le carré de l'accélération tangentielle, l'accélération normale ne servant qu'à conserver la courbe sur l'hypersphère. Les courbes splines sont, comme dans notre approche décrite plus loin, basées sur la formulation d'Hermite. Dans l'espace euclidien, les vitesses initiales et finales servent de conditions aux bornes; sur l'hypersphère de $\mathcal{R}^4$, ce sont les vitesses angulaires qui tiennent ce rôle.

Evaluer le quaternion $q(t)$ qui minimise la courbure est un problème de calcul des variations. Le problème est discrétisé et simplifié en décomposant la fonction à minimiser en un ensemble d'inconnues q_i qui sont des échantillons de la fonction cherchée. Au lieu de minimiser une intégrale, il minimisent donc une somme discrète. Les contraintes sont ajoutées aux critères à l'aide des multiplicateurs de Lagrange.

De même que dans $\mathcal{R}^3$, le tracé d'une spline revient à calculer des échantillons de la spline et à les relier par des segments, la dernière étape de cette méthode consiste à relier entre eux les quaternions q_i grâce à l'interpolation linéaire des quaternions décrite plus haut.

Il faut souligner que la méthode de résolution est très lente puisque basée sur un algorithme de minimisation itératif. Les auteurs reconnaissent que l'interpolation d'une dizaine d'orientations clés nécessite plusieurs minutes.

1.4 Une interpolation basée sur les logarithmes de quaternions

Partant du principe que les distorsions entrainées par la non-commutativité des quaternions dans l'interpolation linéaire sphérique (§ 1.2.3) sont relativement négligeables, nous allons nous baser sur la même hypothèse pour créer des splines d'interpolation sphériques à partir des logarithmes de quaternions. En ce sens, notre solution est proche de celle du paragraphe 1.3.4, excepté que notre paramétrisation est plus simple.

Le principal reproche qui peut être fait aux solutions évoquées aux §1.3.1, 1.3.2 et 1.3.3 est d'éluder le problème de l'interaction. Celle reposant sur un algorithme de minimisation est trop lente. La méthode basée sur les exponentielles de matrice antisymétriques y fait par contre fortement référence. Cependant, l'utilisateur, n'agissant que sur des vecteurs $\mathcal{R}^3$, a du mal à appréhender les conséquences de ses manipulations sur l'interpolation des quaternions.

Nous nous proposons de permettre directement la manipulation de *tangentes sphériques*. Aussi, la méthode d'interpolation choisie est celle d'Hermite. En effet, celle-ci permet de définir une courbe entre deux points en spécificant, en plus de ces deux points, les demi-tangentes aux extrémités.

L'expression d'un point $P_i(u)$ de la courbe peut, comme précédemment, s'exprimer sous forme matricielle :

$$P_i(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} M_{Hermite} \begin{pmatrix} P_i \\ D_i \\ P_{i+1} \\ G_{i+1} \end{pmatrix} \tag{1.17}$$

où P_i et P_{i+1} sont des vecteurs contenant les coordonnées des deux points entre lesquels se fait l'interpolation, D_i étant la tangente à droite en P_i et G_{i+1} la tangente à gauche en P_{i+1} (figure 1.10). La matrice $M_{Hermite}$ est donnée dans le paragraphe 1.3, page 25, sa construction est détaillée en annexe B. Une manière simple de définir les tangentes d'un point P_i est d'utiliser la méthode des splines cardinales, à savoir $\alpha(P_{i+1} - P_{i-1})$ pour la tangente à droite et $\alpha(P_{i+1} - P_{i-1})$ pour la tangente à gauche.

De la même façon, nous pouvons décrire le morceau de courbe sphérique reliant deux quaternions q_i et q_{i+1}, en utilisant leurs logarithmes et les logarithmes de leurs tangentes

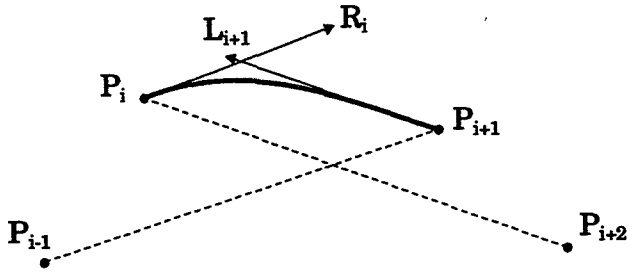


FIG. 1.10 Interpolation d'Hermite

sphériques :

$$q(u) = \exp\left(\begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} M_{Hermite} \begin{pmatrix} \log q_i \\ \log qd_i \\ \log q_{i+1} \\ \log qg_{i+1} \end{pmatrix}\right) \tag{1.18}$$

Le quaternion qg_{i+1} représente la tangente sphérique à gauche en q_{i+1}, le quaternion qd_i représente la tangente sphérique à droite en q_i. On verra par la suite comment calculer les valeurs par défaut de ces tangentes sphériques.

En résumé, pour interpoler deux orientations-clés, il faut calculer les logarithmes de quatre quaternions, interpoler ces logarithmes et retourner aux quaternions par l'exponentielle des quaternions de partie réelle nulle.

1.5 Edition des courbes sphériques

Des solutions ont été apportées dans le paragraphe précédent au problème de la création de courbes lisses interpolant des orientations. Mais qu'en est-il du contrôle sur ces courbes? Est-il possible d'en modifier l'aspect comme le font de nombreux éditeurs de courbes dans $\mathcal{R}^3$? Shoemake, Duff et Pletincks n'ont pas abordé ce problème. Par contre c'est un des thèmes principaux des deux articles de Yahia et Gagalowicz. Leur approche consiste à passer d'une paramétrisation dans $\mathcal{S}^3$ à une paramétrisation dans $\mathcal{R}^3$ (cf §1.3.4). Il suffit alors de manipuler les courbes définies dans $\mathcal{R}^3$ par des techniques interactives décrites dans [CB88]. Toute modification sur la courbe de $\mathcal{R}^3$ entraîne alors automatiquement une modification de la courbe interpolant les orientations.

Cependant, on peut penser que l'utilisateur a du mal à appréhender les conséquences que ses manipulations, dans l'espace cartésien, vont avoir sur la courbe spline sphérique. En effet, la position d'un point de $\mathcal{R}^3$ représente une paramétrisation d'une orientation ; c'est à dire non seulement l'axe de la rotation mais aussi son angle. A partir de là, la modification d'une tangente, par exemple, va entraîner des perturbations de l'interpolation des orientations à ces deux niveaux, sans qu'elles soient vraiment contrôlées.

Pour ces raisons, nous nous proposons de présenter ici une méthode permettant d'utiliser ce que l'on appelle des tangentes sphériques. Nous allons voir que ces dernières per-

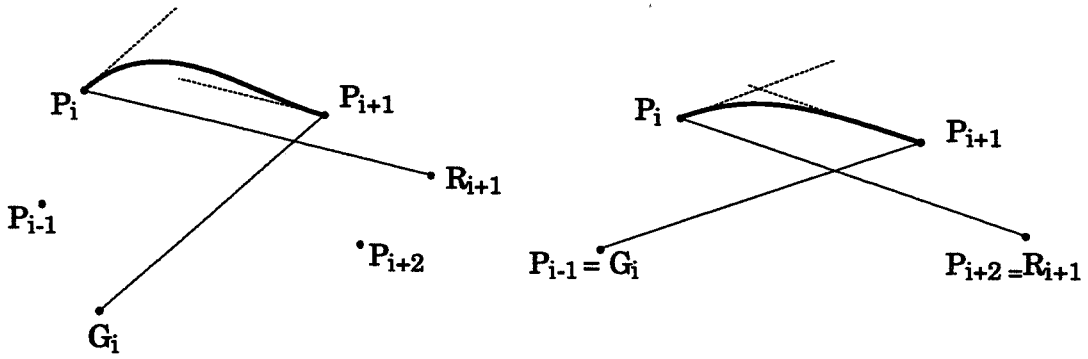


FIG. 1.11 Utilisation de points supplémentaires pour spécifier les tangentes (a). Valeurs par defaut (b).

mettent de modifier la courbe d'interpolation des orientations de façon intuitive pour l'animateur. Ces tangentes sphériques ne représentent pas réellement la tangente, au sens cinématique du terme, mais plutôt une aide visuelle pour l'interaction.

Si les orientations-clés permettent à l'utilisateur de spécifier les caractéristiques essentielles du mouvement, les tangentes sphériques lui offrent la possibilité d'effectuer des réglages plus fin, qui donneront la touche finale au mouvement.

Nous allons commencer par rappeler les solutions interactives pour l'édition de courbes splines dans $\mathcal{R}^3$. Ensuite, nous définirons ce qu'est une tangente sphérique, puis enfin, les moyens de manipuler ces courbes sphériques.

1.5.1 Une méthode d'édition de courbes splines dans $\mathcal{R}^3$ à l'aide des tangentes

Nous allons détailler ici une technique d'interaction pour des splines de $\mathcal{R}^3$ présentée dans [CB88]. Les courbes manipulées sont des splines cardinales (courbes par morceaux). La solution proposée est d'introduire, pour chaque couple de points consécutifs P_i et P_{i+1}, deux points de contrôle supplémentaires G_i et R_{i+1} qui, indirectement, définissent les tangentes aux deux extrémités du morceau de courbe ayant G_i, P_i, P_{i+1} et R_i pour points de contrôle (figure 1.11.a). Au départ, ces nouveaux points sont placés de telle sorte que la nouvelle courbe coïncide avec la courbe initiale : $G_i = P_{i-1}$ et $R_{i+1} = P_{i+2}$ (figure 1.11.b)

Il est alors possible de déplacer ces points fictifs de manière à modifier les tangentes en chaque point. Outre les points de contrôle initiaux, les tangentes et la courbe elle-même sont visualisées à l'écran. Evidemment, il est possible de déplacer interactivement ces tangentes. Les répercussions sur la courbe sont bien sûr visualisées au fur et à mesure que l'on déplace les tangentes. En plus de la possibilité de modifier les positions des points

de contrôle, les facilités suivantes sont permises :

- Les tangentes à gauche G_i et à droite R_i d'un point P_i peuvent être astreintes à rester colinéaires, et ceci de manière à conserver la continuité de la courbe. Annuler cette contrainte permet de créer des discontinuités.
- On peut modifier le module d'une tangente de façon à créer un effet de tension.
- En changeant la direction de la tangente, on peut introduire un effet de biais (décalage de la courbure par rapport au point de contrôle).

Notons qu'une autre solution intéressante a été présentée dans [KB84]. Ici, l'utilisateur manipule trois paramètres de la courbe spline : la tension, le biais et la continuité. Cependant, comme il est souligné dans [CB88], les effets de ces trois paramètres peuvent être recréés en manipulant les tangentes.

1.5.2 Edition de courbes splines sphériques

Nous utilisons le même principe de manipulation, mais cette fois, au lieu de déplacer des points et des tangentes dans $\mathcal{R}^3$, on va manipuler les tangentes sur une sphère de dimension 3. D'autre part, on va maintenant se servir de l'interpolation d'Hermite. En effet, pourquoi créer des points invisibles pour simuler les tangentes alors que l'interpolation d'Hermite (1.12) permet immédiatement de créer la courbe reliant deux points, connaissant les tangentes en ces deux points? Il suffit d'initialiser ces tangentes à partir des deux points voisins du couple de points considérés, en utilisant la définition des tangentes pour les splines de Catmull-Rom (splines cardinales avec $\alpha = \frac{1}{2}$) :

- $G_i = \frac{1}{2}(P_{i+1} - P_{i-1})$ pour la demi-tangente à gauche au point P_i
- $D_i = \frac{1}{2}(P_{i-1} - P_{i+1})$ pour la demi-tangente à droite.

Visualisation des orientations

Les quaternions peuplent un espace de dimension quatre. Or, la dimension trois est déjà difficilement représentable sur un écran de dimension deux. La solution que nous avons choisie est de privilégier un point de l'objet et d'afficher la trajectoire de ce point soumis aux orientations successives. Cela permet de descendre d'une dimension, puisque l'image d'un vecteur par un quaternion est un élément de $\mathcal{R}^3$. L'ensemble des positions de ce point particulier soumis aux orientations successives se situe sur une sphère de $\mathcal{R}^3$ (figure 1.12).

Prenons l'exemple d'un polyèdre dont on désire interpoler les orientations successives. Il faut commencer par sélectionner l'un des sommets de ce polyèdre et en récupérer les coordonnées dans son repère local. Soulignons que les orientations-clés décrivent impérativement l'orientation du repère local. Ensuite, on peut visualiser la trajectoire de ce point en lui appliquant les rotations interpolées. Dans ce qui suit, nous appelons P_i, l'image par l'orientation q_i du point sélectionné de l'objet à animer.

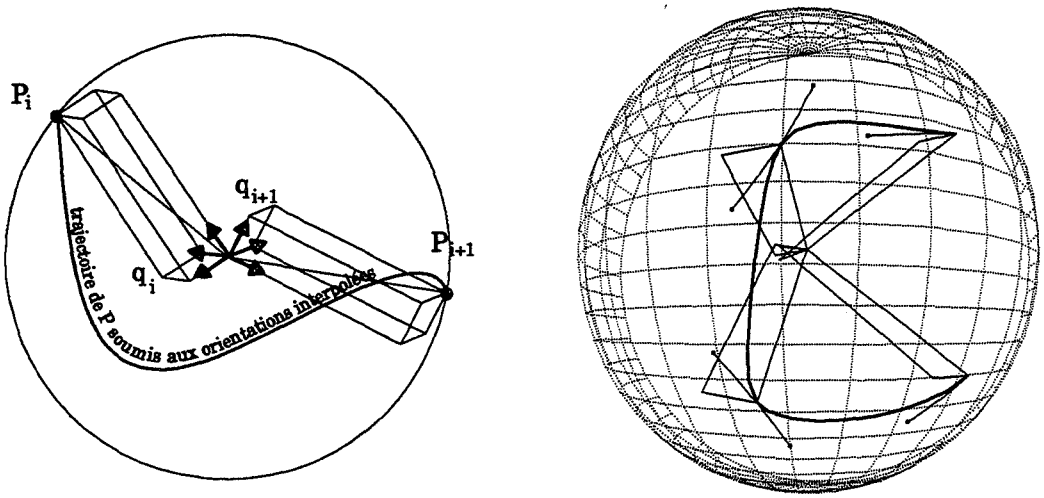


FIG. 1.12 Représentation des orientations

Définition des tangentes sphériques

Partant de la définition des tangentes au sens des splines cardinales, et en raisonnant en terme de quaternions, on définit la *demi-tangente sphérique à gauche* (notée qg_i) du quaternion q_i de la façon suivante :

$$qg_i = (q_{i+1}^{-1} q_{i-1})^{\frac{1}{2}}$$

De même, la *demi-tangente sphérique à droite* qd_i est définie par :

$$qd_i = (q_{i-1}^{-1} q_{i+1})^{\frac{1}{2}}$$

Ces tangentes sont représentées par des quaternions. Par exemple,lLa tangente qd_i représente la moitié de la rotation menant de l'orientation q_{i-1} à l'orientation q_{i+1}.

En fait, pour rester compatible avec les splines que l'on veut utiliser, nous allons définir ces tangentes en termes de logarithmes de quaternions. Ainsi, le logarithme de la tangente à gauche est :

$$\log qg_i = \frac{1}{2}(\log q_{i-1} - \log q_{i+1})$$

Celui de la tangente à droite est

$$\log qd_i = \frac{1}{2}(\log q_{i+1} - \log q_{i-1})$$

En ce qui concerne la visualisation des tangentes telles qu'elles sont représentées sur la partie droite de la figure (1.12), il suffit d'appliquer au point P_i le quaternion définissant la tangente. On approxime l'arc de cercle symbolisant cette tangente par une suite de points, reliés par des segments. Dans le cas de la tangente à gauche par exemple, chacun de ces points est l'image par une fraction de cette tangente ($u \log qg_i$, u variant entrer 0 et 1) du point P_i.

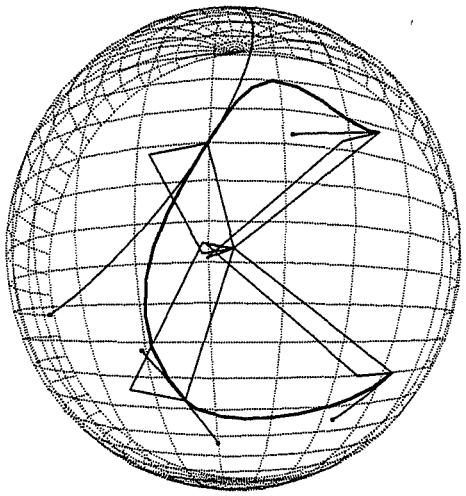


FIG. 1.13 Augmentation de l'amplitude d'une tangente

Edition des tangentes sphériques

Disposant maintenant d'une technique pour visualiser les tangentes sphériques, nous pouvons envisager de les modifier interactivement. Pour cela, on affiche à l'extrémité de chaque tangente un symbole (□) que l'animateur va pouvoir sélectionner avec la souris et déplacer en fonction de deux contraintes que nous allons définir. Ces contraintes restreignent l'animateur à modifier soit le module, soit la direction d'une tangente. Le choix entre ces deux options est fait par menu. Concernant l'interaction 3D, de plus amples détails seront fournis dans le chapitre suivant.

Modification du module. Cette opération consiste à modifier l'amplitude de la tangente, c'est à dire l'angle de l'arc sur $\mathcal{S}^3$ représentant cette tangente. Un étirement augmente l'arrondi de la courbe (figure 1.13). Un rétrécissement entraîne une courbe plus pointue (figure 1.14).Ces figures sont à comparer avec la partie droite de la figure (1.12) où les tangentes ont leurs valeurs par défaut.

La tangente est symbolisée par un arc de cercle passant par P_i et de centre O (voir figure 1.15). Changer le module de cette tangente revient à modifier la longueur de cet arc, tout en restant sur le cercle de centre O passant par P_i. Il faut donc contraindre les mouvements de la souris à rester sur ce cercle.

Pour cela, on commence par déterminer dans le repère de la scène, l'équation de la droite passant par les coordonnées de la souris et orthogonale au plan de l'écran. On calcule ensuite l'intersection I entre cette droite et le plan contenant la tangente et enfin, on projette ce point sur le cercle, ce qui fournit la nouvelle position de l'extrémité de cette tangente (□).

L'expression de la nouvelle tangente qg'_i (dans le cas d'une tangente à gauche) est alors

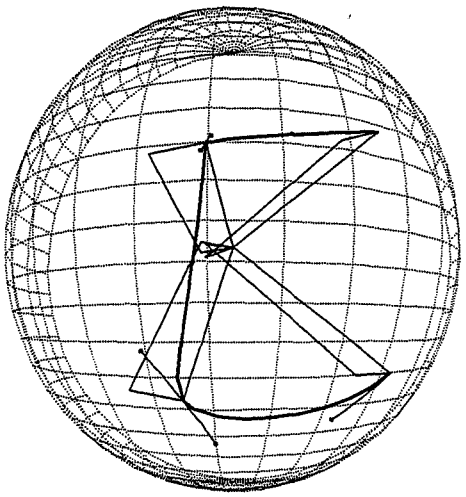


FIG. 1.14 Diminution de l'amplitude d'une tangente

donnée par :

$$\log qg_i' = \frac{\theta_i'}{\theta_i} \log qg_i$$

où θ_i' (resp. θ_i) désigne l'angle de l'arc de cercle de centre O reliant P_i à l'extrémité de la nouvelle tangente qg_i' (resp. de l'ancienne tangente qg_i). L'axe de rotation de la tangente est invariant, seul l'angle, donc la longueur de l'arc, change.

Modification de la direction. Dans ce cas, on offre la possibilité de faire tourner l'arc de cercle représentant la tangente à un quaternion q_i autour de P_i. Les figures 1.18 et 1.19 donnent un aperçu des modifications engendrées sur la courbe par deux rotations différentes de la tangente initiale. On peut constater l'effet de biais généré.

Si l'angle de cette rotation varie entre 0 et 2π, la tangente va décrire une calotte autour du point considéré. Le rayon de cette calotte est calculé aisément en fonction de l'angle de l'arc représentant la tangente. L'extrémité de la tangente va être contrainte à se déplacer sur cette calotte, ou plus précisément, sur le cercle délimitant le bord de cette calotte (voir figure 1.16).

Comme auparavant, on détermine l'équation de la droite passant par la souris et orthogonale au plan de l'écran. L'intersection entre cette dernière et le plan de la calotte nous donne un point I que l'on projette sur le petit cercle. C'est la nouvelle position de l'extrémité de la tangente.

Pour trouver l'expression de la tangente sphérique modifiée, examinons le cas de la tangente à gauche. On appelle G_i l'extrémité de la tangente initiale et G_i' l'extrémité de la tangente modifiée. Lors de la manipulation décrite ci-dessus, le point G_i a été transformé en G_i' par une rotation d'angle ϕ (figure 1.17). La tangente à gauche initiale décrit un arc

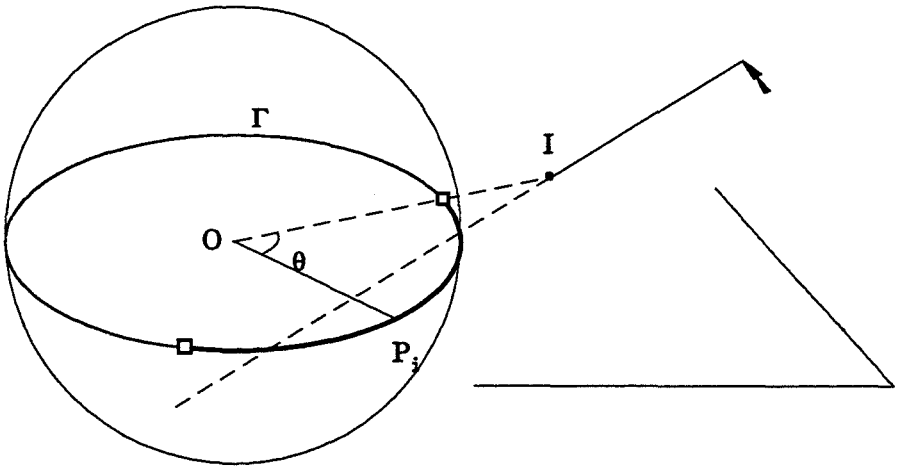


FIG. 1.15 Modification interactive du module de la tangente

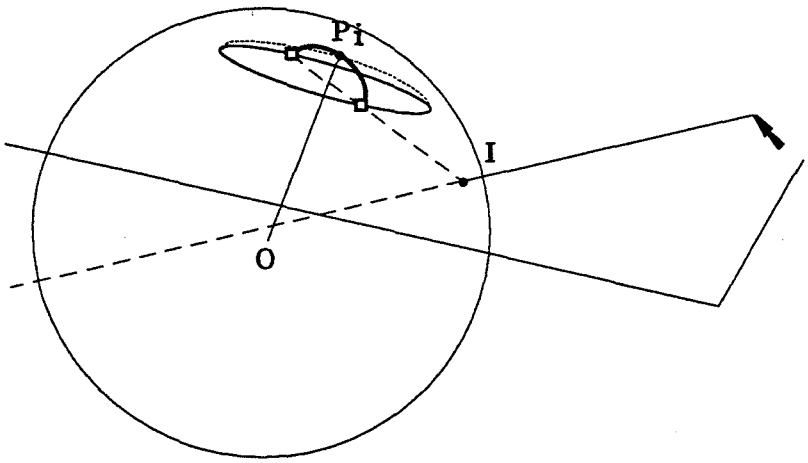


FIG. 1.16 Modification interactive de la direction de la tangente

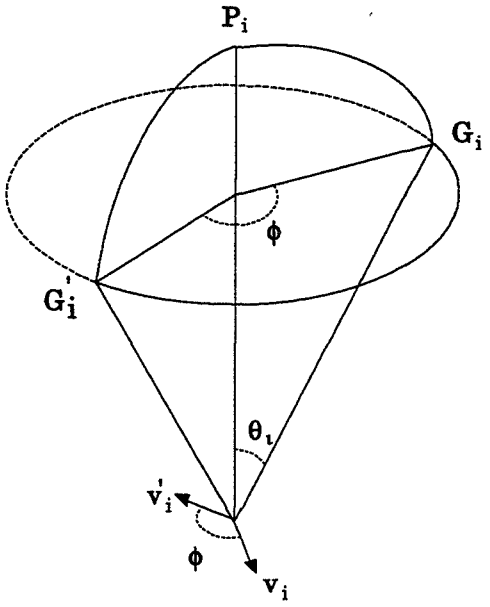


FIG. 1.17 Calcul de la nouvelle tangente sphérique

de cercle d'angle $\theta_i = 2\|\log qg_i\|$ autour du vecteur :

$$v_i = \frac{\log qg_i}{\|\log qg_i\|}$$

La nouvelle tangente sphérique décrit un arc de longueur inchangée θ_i autour du vecteur v_i' qu'il nous faut déterminer. v_i' est le vecteur v_i auquel a été appliquée une rotation d'axe OP_i et d'angle ϕ. Celle-ci peut-être représentée par le quaternion :

$$q_r = e^{\frac{\phi}{2}\frac{\overrightarrow{OP_i}}{\|\overrightarrow{OP_i}\|}}$$

Le logarithme du quaternion qg_i' représentant la nouvelle tangente à gauche sera donc

$$\log qg_i' = \frac{\theta_i}{2} v_i'$$

avec

$$[0, v_i'] = q_r [0, v_i] q_r^{-1}$$

Discontinuités. Dans les exemples précédents, il y avait un couplage entre la demi-tangente à gauche et la demi-tangente à droite. En d'autres termes, les deux tangentes sphériques étaient toujours situées sur le même cercle. On peut cependant éditer chaque tangente séparément pour créer des discontinuités. Un exemple est donné sur la figure 1.20 où les tangentes à droite et à gauche ont été déplacées indépendamment l'une de l'autre. Elles ne sont plus sphériquement colinéaires.

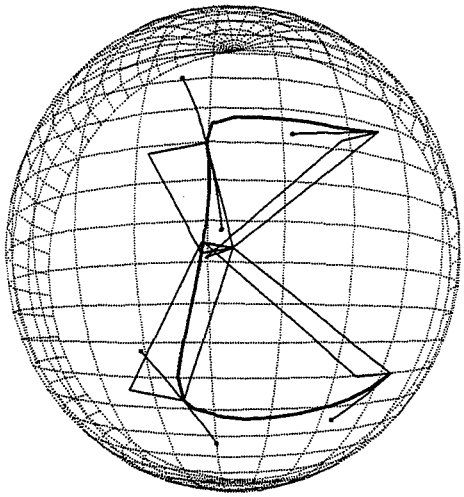


FIG. 1.18 Rotation de la tangente

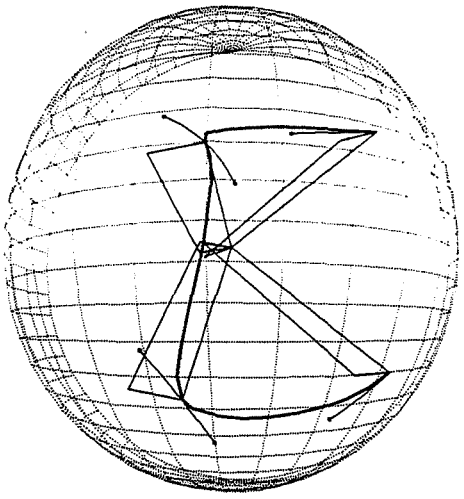


FIG. 1.19 Rotation accentuée de la tangente

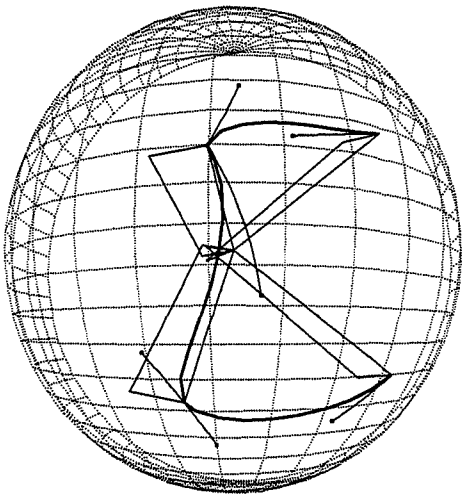


FIG. 1.20 Tangentes non sphériquement colinéaires

1.6 Interpolation des orientations d'un bras articulé

Maintenant que nous avons à notre disposition un outil efficace pour créer et manipuler des courbes lisses dans l'espace des orientations, il est envisageable de l'utiliser pour l'édition de la trajectoire d'un bras articulé.

Un bras articulé est une suite d'éléments connectés entre eux par des articulations. On se place dans le cas où ces articulations sont rotoïdes, c'est à dire qu'elles ne disposent que de degrés de liberté en rotation. L'orientation d'un élément i du bras est définie par rapport à celle de l'élément $i-1$ par un quaternion q_i^{local}.

L'orientation dans un référentiel fixe est donnée par :

$$q_i^{ref} = q_{i-1}^{ref} q_i^{local}$$

Il y a deux manières de considérer l'animation d'un bras articulé ; la première est fondée sur la cinématique inverse, la deuxième sur l'interpolation d'orientations. Nous commençons par décrire ces deux approches en considérant les avantages et inconvénients de chacune d'elles au niveau de la qualité du mouvement obtenu mais aussi des possibilités d'interaction.

La première démarche consiste à ne prendre en compte que la trajectoire de l'élément terminal. Cette trajectoire est définie simplement par une courbe interpolant dans $\mathcal{R}^3$ les positions-clés successives de l'extrémité du bras articulé. Ensuite, on utilise un algorithme de cinématique inverse pour déterminer les positions interpolées des éléments intermédiaires en fonction de la position interpolée de l'extrémité.

L'inconvénient de cette méthode est que l'on ne contrôle absolument pas les trajectoires des articulations intermédiaires. Ces trajectoires peuvent laisser apparaître des discontinuités, même si la trajectoire terminale est une courbe lisse. De plus, une faible variation

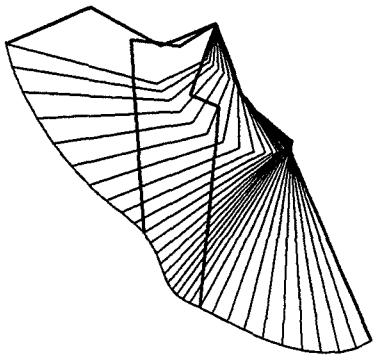


FIG. 1.21 Utilisation de la cinématique inverse. Les positions-clés sont tracées en traits gras; les positions intermédiaires, calculées par cinématique inverse, sont en traits plus fins.

de la trajectoire de l'élément terminal peut entraîner de fortes variations dans celles des éléments intermédiaires, et inversement.

Pour illustrer ce phénomène référons-nous à la figure (1.21) où les positions successives d'un bras articulé ont été définies en quatre instants. Le mouvement global est obtenu en interpolant les positions de l'élément terminal, puis en appliquant un algorithme de cinématique inverse à chaque image interpolée[4].

Evidemment, les positions intermédiaires du bras ne respectent pas les positions-clés spécifiées. La méthode en question n'est pas censée le faire. D'autre part, on se rend compte dans la pratique que les éléments intermédiaires du bras (exceptée l'extrémité) suivent une trajectoire irrégulière. On constate, et particulièrement au début du mouvement (le sens du mouvement est de la droite vers la gauche), que même si l'élément terminal du bras se déplace beaucoup, les autres membres subissent peu de variations.

L'avantage de cette solution est que l'on peut facilement modifier l'allure de l'interpolation de l'élément terminal, par exemple avec les techniques présentées au §1.5.1.

La deuxième solution, présentée sur la figure 1.22, consiste à interpoler indépendamment les orientations de chaque élément du bras à l'aide des courbes splines sphériques introduites précédemment. L'orientation d'un élément décrit une spline sphérique dans le repère de l'élément précédent. La trajectoire finale est obtenue en replaçant la trajectoire locale de chaque élément dans le repère interpolé de l'élément précédent.

Le résultat est une trajectoire qui respecte les orientations-clés de chaque partie du bras. Concernant l'interaction, on peut envisager bien sûr de modifier les trajectoires locales de chacun des membres du bras (avec les tangentes sphériques). Par contre, l'édition interactive de la trajectoire de l'extrémité du bras semble difficile, puisque justement, elle dépend des trajectoires locales.

Notre objectif est donc de trouver une solution alliant les avantages des deux méthodes exposées ci-dessus. L'utilisateur doit pouvoir éditer interactivement non seulement les

[4]L'algorithme de cinématique inverse utilisé est exposé dans [Han91] ainsi que dans le chapitre suivant.

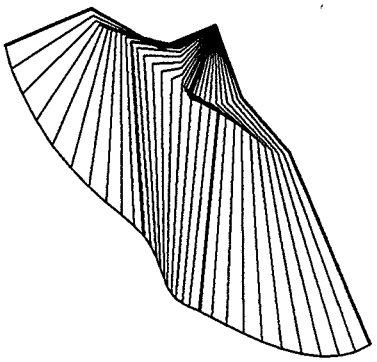


FIG. 1.22 Interpolation des orientations de chaque élément du bras

splines décrivant l'orientation locale de chaque élément mais surtout la courbe globale représentant la trajectoire de l'extrémité du bras. D'autre part, les orientations de chaque élément du bras articulé doivent rester des splines sphériques.

Typiquement, dans le cas d'un bras articulé, nous sommes confrontés à un phénomène de mouvements relatifs, la trajectoire de chaque articulation dépendant des orientations des articulations précédentes. Pour clarifier les idées, nous allons commencer par regarder ce qui se passe pour des mouvements relatifs en position.

1.6.1 Trajectoires relatives en position

On considère un ensemble d'objets i se déplaçant sur des trajectoires $R_j^i(u)$ définies par une interpolation d'Hermite entre deux points P_j^i (muni d'une tangente à droite D_j^i) et P_{j+1}^i (muni d'une tangente à gauche G_{j+1}^i).

Regardons maintenant ce qui se passe si la trajectoire de chaque objet i est relative à un repère lié à l'objet d'indice $i-1$. En d'autres termes, comment calculer les trajectoires absolues A_j^i (c'est à dire par rapport à un référentiel fixe) d'un objet i en fonction des trajectoires relatives des objets d'indices inférieurs. La première solution qui vient à l'esprit consiste à évaluer les courbes relatives $R_j^i(u)$ pour chaque valeur de u et d'en faire la somme:

$$A_j^n(u) = \sum_{i=1}^{n} R_j^i(u)$$

avec

$$R_j^i(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} (M_{Hermite}) \begin{pmatrix} P_j \\ D_j \\ P_{j+1} \\ G_{j+1} \end{pmatrix}$$

En fait, on remarque aisément que la trajectoire absolue, qui est la composition d'un

ensemble de trajectoires relatives, peut s'évaluer directement de la façon suivante :

$$A_j^n(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} (M_{Hermite}) \begin{pmatrix} \sum_{i=1}^n P_j^i \\ \sum_{i=1}^n D_j^i \\ \sum_{i=1}^n P_{j+1}^i \\ \sum_{i=1}^n G_{j+1}^i \end{pmatrix}$$

Si on considère que les P_j^i, G_j^i et D_j^i représentent des données relatives, alors on peut définir :

- des points absolus :

$$P_j^n = \sum_{i=1}^{n} P_j^i$$

- des demi-tangentes à gauche absolues :

$$G_j^n = \sum_{i=1}^{n} G_j^i \tag{1.19}$$

- des demi-tangentes à droite absolues :

$$D_j^n = \sum_{i=1}^{n} D_j^i \tag{1.20}$$

On peut se poser la question de savoir comment répercuter sur les courbes relatives les modifications apportées à la courbe résultante ; c'est à dire, comment modifier les tangentes relatives après avoir modifié les tangentes absolues. On peut par exemple envisager de faire subir à une des tangentes relatives la même transformation que celle subie par la tangente absolue. L'inconvénient est qu'une seule courbe supporte tous les effets de la modification.

La solution est de répartir la modification subie par la tangente absolue sur toutes les tangentes relatives. Un changement de module de la tangente absolue entraîne une modification équivalente de chacune des tangentes relatives. Une rotation de la tangente absolue est repercutée par une rotation identique de chacune des tangentes sphériques. On vérifie trivialement que les relations (1.19) et (1.20) restent valides. Au niveau de l'interaction, les opérations de rotations ou de modification du module peuvent, bien entendu, être combinées.

1.6.2 Trajectoires relatives en orientation

Revenons maintenant à l'animation de notre bras articulé. Chaque bras articulé i possède un ensemble d'orientations-clés locales $\log q_j^i$ et de tangentes elles aussi locales $\log qg_j^i$ et $\log qd_j^i$ entre lesquelles on sait interpoler par des splines sphériques $\log q_j(u)$. Comme ci-dessus, nous définissons des tangentes sphériques absolues à chaque orientation-clé j :

$$\log qg_j^n = \sum_{i=1}^{n} \log qg_j^i$$

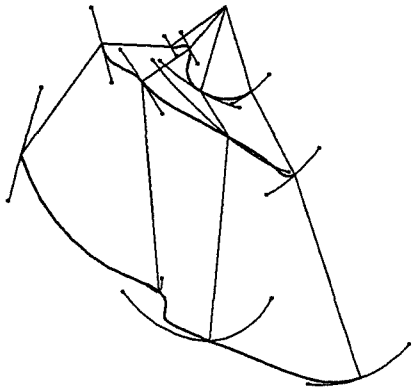


FIG. 1.23 Tangentes absolues sur un bras articulé

$$\log qd_j^n = \sum_{i=1}^{n} \log qd_j^i$$

De même que les tangentes relatives, les tangentes absolues peuvent être représentées (figure 1.23).

L'animateur a alors la possibilité de manipuler interactivement ces tangentes absolues. Un changement d'amplitude d'une tangente absolue entraîne un changement identique de chacune des tangentes relatives dont elle dépend (figure 1.24). De même, une rotation d'une tangente absolue entraîne une rotation équivalente des tangentes relatives (figure 1.25). Ici, seule une demi-tangente a subi une rotation de manière à créer un effet de discontinuité.

A nouveau, il est important de souligner que ces tangentes absolues ne sont pas de vraies tangentes, au sens cinématique du terme; simplement, elles offrent un support visuel à l'utilisateur.

En particulier, l'effet de translation auquel est soumis chaque articulation i du fait du déplacement des éléments d'indice inférieur n'est pas pris en compte dans les tangentes absolues. Plus un élément est proche de la base du bras articulé (plus son indice est faible), plus son bras est long et plus la rotation qu'il subit engendre un grand déplacement à l'extrémité du bras. Ce qui veut dire que la tangente relative de cet élément dispose d'une influence accrue sur les éléments d'indice supérieur. Pour pallier ces effets, on définit en fait les tangentes absolues ainsi :

$$\log qg_j^n = \sum_{i=1}^{n} \lambda_i \log qg_j^i$$

où λ_i désigne la somme de la longueur de tous les bras d'indice inférieur à i.

Evidemment, comme on peut le constater sur les figures, les tangentes sphériques "collent" de moins en moins avec la courbe d'interpolation quand on s'éloigne de l'origine. L'expérience montre que cet outil se montre néanmoins fort utile dans un contexte interactif.

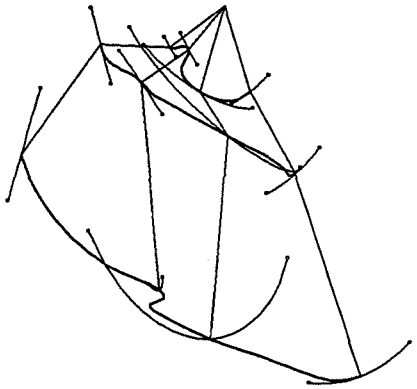


FIG. 1.24 Modification du module d'une tangente absolue

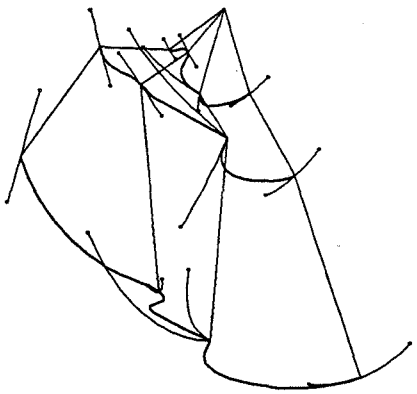


FIG. 1.25 Modification de la direction d'une tangente absolue

1.7 Conclusion

Nous avons présenté des techniques permettant d'interpoler des orientations. Cinq méthodes, fondées sur différents interpolants, ont été étudiées. Les trois premières utilisent des constructions géométriques, la suivante effectue une minimisation de la courbure dans $\mathcal{S}^3$ et la dernière se sert d'une paramétrisation de l'espace des quaternions dans $\mathcal{R}^3$. Nous distinguons donc les trois approches suivantes :

Décomposition géométrique. Cette décomposition peut être exacte dans le cas des courbes de Bézier ou approchée dans les autres cas. Ces constructions utilisent une définition géométrique récursive des interpolants. Les temps de calculs peuvent donc être relativement importants. Enfin, aucun des trois auteurs ne parle d'utiliser sa méthode dans un environnement interactif [Sho85, Duf86, Ple89].

Minimisation de la courbure L'idée d'appliquer directement dans $\mathcal{S}^3$ les définitions usuelles des splines euclidiennes fournit des résultats très semblables aux nôtres (à en juger par les illustrations figurant dans l'article concerné). La lenteur de la résolution interdit cependant toute application interactive.

Paramétrisation de $\mathcal{S}^3$. La paramétrisation utilisant les exponentielles de matrices antisymétriques permet d'éditer les trajectoires d'orientations avec des techniques interactives éprouvées appliquées à $\mathcal{R}^3$. Cependant, comme cela a été démontré, le paramétrage n'est pas exact. Mais surtout, l'utilisateur a du mal à anticiper les conséquences de ses manipulations dans $\mathcal{R}^3$ sur les orientations interpolées.

La solution que nous proposons est du même type que la précédente. Cependant, comme dans le cas précédent, le paramétrage par les logarithmes de quaternions n'est pas exact. Les exemples que nous avons proposés montrent qu'il est malgré tout suffisant.

D'autre part, l'utilisation de l'interpolation d'Hermite permet d'implanter de manière simple et efficace l'utilisation des tangentes sphériques. L'emploi de ces tangentes sphériques offre à l'utilisateur un confort inégalé pour la manipulation des trajectoires d'orientation. C'est à la fois rapide et intuitif, puisqu'il voit en temps réel les modifications engendrées sur la trajectoire des orientations par le déplacement d'une tangente. Notons que l'on aurait pu adapter cette technique interactive à l'une des trois interpolations ci-dessus (fondée sur une décomposition géométrique), mais au détriment peut-être de la rapidité de la réponse.

Enfin, une alternative intéressante aurait été d'utiliser les splines – controlée par les paramètres de tension, biais et continuité – de Kochanek et Bartels. Dans de cas, au lieu d'ajuster l'allure des courbes à l'aide des tangentes sphériques, l'animateur aurait a sa disposition trois potentiomètres permettant de régler les valeurs, en chaque orientation-clés, des trois paramètres indiqués ci-dessus. Cette extension ne pose aucun problème

théorique mais des développements en ce qui concerne les moyen d'interaction.

Nous n'avons pas abordé ici le problème du contrôle des positions. Une partie du chapitre 4 y sera consacré.

Tout ceci a été appliqué à l'animation d'un bras articulé, et permet d'obtenir des trajectoires lisses, non seulement à l'extrémité du bras, mais aussi en chacun des segments le constituant.

Dans le chapitre 4, nous montrerons comment sont intégrées les techniques de contrôle d'orientation présentées ici avec des techniques plus classiques d'interpolation de positions.

Chapitre 2

L'interface

Dans le chapitre précédent, des solutions permettant le contrôle interactif de courbes d'interpolation ont été présentées. Nous y avons montré le besoin de techniques interactives de haut niveau, comme par exemple l'allongement d'une tangente sphérique. Toutes ces opérations interactives tri-dimensionnelles utilisent un noyau de base dont nous allons donner ici une description plus détaillée. Quoique destiné initialement à résoudre un problème spécifique, ce noyau se révèle en fait particulièrement bien adapté à des applications de saisie tri-dimensionnelles plus générales.

En deux dimensions, un système interactif se doit d'assurer quatre tâches essentielles qui sont (voir [FvDFH90]):

- Le positionnement, qui consiste à fournir une coordonnée (x, y) à partir d'un périphérique.
- La sélection, qui permet de choisir un élément parmi ceux affichés.
- La saisie de texte.
- La saisie de valeurs numériques.

Nous ne nous intéresserons en fait qu'aux deux premières de ces tâches qui, évidemment, se révèlent bien plus compliquées à assurer en dimension trois. La première raison est qu'il est difficile d'appréhender la notion de profondeur et les relations entre objets. La deuxième vient du fait que la grande majorité des périphériques de saisie se limitent à des déplacements plans. Néanmoins, concernant ce dernier point, des solutions ont été apportées grâce au développement de divers périphériques disposant de trois degrés de liberté [SD91a]. Leur facilité d'emploi, du point de vue de l'utilisateur, reste à notre avis à démontrer. En dimension trois, une autre fonctionnalité importante est nécessaire: la spécification des orientations, c'est à dire la possibilité d'appliquer des rotations aux objets.

Dans ce chapitre, nous exposons les solutions retenues afin de réaliser une interface tri-dimensionnelle à l'aide d'un périphérique 2D classique telle qu'une souris.

2.1 Techniques précédentes

Pour remédier à la différence de dimensionnalité entre le périphérique de saisie et la scène, il est nécessaire d'imposer des contraintes aux mouvements afin de faire correspondre les déplacements 2-D de la souris à des mouvements 3-D. Cela revient à limiter à deux, voire à un, le nombre de degrés de liberté manipulés à la fois.

Une des solutions fréquemment employée, et qui découle directement des techniques de la dimension deux, consiste à afficher plusieurs projections orthogonales de la scène (haut, face, côté). La souris permet alors de positionner un objet par rapport aux deux axes orthogonaux au plan de la projection ou de l'orienter relativement à l'axe perpendiculaire à l'écran.

Une autre possibilité consiste à utiliser la projection à l'écran du trièdre symbolisant le référentiel de la scène [NO86]. Ces trois axes représentent les directions le long desquelles les mouvements de la souris seront effectifs (figure 2.1). Le mouvement est limité à un axe à la fois et cet axe est celui qui se trouve correspondre le mieux à la direction de déplacement de la souris. En quelque sorte, ce système essaie de deviner le mouvement souhaité par la personne qui manipule.

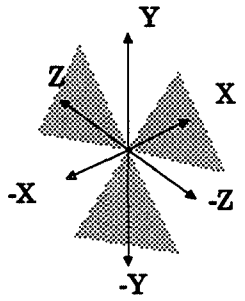


FIG. 2.1 Utilisation de directions privilégiées

Les approches les plus récentes utilisent en fait des contraintes dépendantes du contexte. Ce sont les axes des repères locaux des objets manipulés qui déterminent les directions du mouvement [NO86]. Cette solution est généralisée dans [Ove89] et [Nie91] avec l'introduction de la notion de curseur-3D. Un curseur est un trièdre quelconque positionné par l'utilisateur et dont les axes définissent les directions privilégiées des translations ou les axes privilégiés permettant d'effectuer les rotations. Parmi les trois axes du trièdre, l'utilisateur peut en sélectionner un ou deux qui définissent des sous-espaces de $\mathcal{R}^3$ dans lesquels sont contraints les mouvements.

2.2 Principe de l'interface

L'interface que nous présentons maintenant s'inspire sur quelques points de certains principes énoncés ci-dessus. En particulier, nous limitons les déplacements à une ou deux dimensions de manière à assurer la possibilité de correspondance entre la souris et les

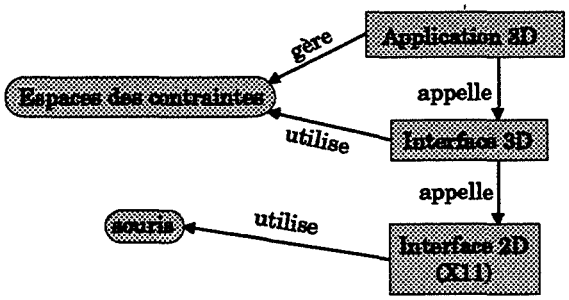


FIG. 2.2 Relations entre l'interface et l'application

déplacements dans la scène. Ensuite, nous posons comme règle que la position de la souris et celle de l'objet déplacé doivent coïncider. En guise d'illustration, une contrainte de mouvement peut être de restreindre les mouvements à des translations dans un plan; au cours des mouvements de la souris, l'objet manipulé se déplace dans ce plan tout en accompagnant les déplacements de la souris.

D'un autre côté, l'utilisation fréquemment faite du contexte de la scène pour spécifier les contraintes de mouvement nous semble limitatif. On a vu ci-dessus que le contexte servait à définir a priori des règles de l'interface. Or ce contexte est une structure sans rapport avec le système d'interfaçage. Il est en effet constitué, entre autres, de l'ensemble des objets de la scène et en particulier des repères locaux de ces derniers qui servent à spécifier les directions privilégiées des mouvements.

C'est sans doute une approche qui simplifie les choses quand il ne s'agit que de translater ou de faire tourner les objets de la scène. Pour certaines tâches interactives dont nous avons besoin, par exemple appliquer une rotation à une tangente sphérique – qui ne dispose pas de repère local approprié – ce système présente des faiblesses.

Notre choix est donc le suivant : Nous laisserons à l'application le soin de déterminer les caractéristiques des contraintes de mouvement. C'est à elle d'en gérer les caractéristiques, par exemple la sélection des axes de rotation ou des axes de translation et d'en préciser ensuite les caractéristiques au système réalisant l'interface 3-D. Les relations entre les diverses parties du système sont mises en évidence sur la figure 2.2.

Quant aux contraintes de mouvement, nous ne voulons pas nous limiter aux mouvements standard qui sont les translations en une et deux dimensions et les rotations avec un ou deux degrés de liberté. En toute généralité, tout sous-espace de $\mathcal{R}^3$ doté de un ou deux degrés de liberté permet de définir une contrainte de mouvement. Dans notre système, tous sous-espace paramétrable par deux variables permet de représenter une contrainte de mouvement : droite, plan, cercle, sphère, courbe paramétrique, surface bi-paramétrique. Une contrainte est donc définie par des caractéristiques géométriques et un ou deux paramètres.

La gestion des interactions pour une application utilisant notre interface 3-D se décompose de la façon suivante:

- L'application se charge de définir et de gérer les caractéristiques définissant la con-

trainte de mouvement à partir des spécifications que l'utilisateur fait, via la souris ou le clavier. Par exemple, celui-ci va sélectionner deux axes et indiquer qu'il veut effectuer une translation dans un plan.

- L'utilisateur sélectionne un objet et l'application se charge d'indiquer à l'interface (au moyen de passage de paramètres) les caractéristiques du sous-espace contraignant les déplacements à venir.

- L'interface gère alors seule les mouvements de la souris. A chaque déplacement significatif[1] de celle-ci, elle détermine le déplacement à faire subir à l'objet pour qu'il suive la souris tout en respectant la contrainte de mouvement

- Le processus s'arrête quand l'utilisateur relache la souris.

A chacun de ces déplacements significatifs, la mise à jour des coordonnées de l'objet déplacé (position, orientation, etc...) et de l'affichage sont à la charge de l'application. Ceci est réalisé au moyen d'une fonction, définie dans l'application, chargée d'effectuer ces tâches. Cette fonction est exécutée à la demande de l'interface, à chaque mouvement significatif.

A chaque déplacement de la souris, l'interface détermine de son côté l'intersection entre l'espace des contraintes et la souris. Il en déduit les variations que subissent la ou les variables qui paramétrisent la contrainte. Ce sont les variations des paramètres qui permettent à l'application de réaliser les mises à jour dont nous parlions.

On le voit, l'interface a deux fonctions principales : la première est de calculer l'équation de la droite *œil-curseur*. Il s'agit de déterminer, à partir des coordonnées (x, y) de la souris, cette droite passant par l'œil et la souris, et ceci dans le repère du monde. Nous verrons cela dans la partie qui vient.

Cette droite étant connue, la deuxième opération consiste à déterminer l'intersection entre elle et l'espace de la contrainte. Dans le paragraphe 2.3 nous énumérerons les types de contraintes et en décrirons la paramétrisation. Enfin, dans le paragraphe 2.4 seront détaillés les calculs de l'intersection entre la droite *œil-curseur* et les différents types de contraintes. Souvent, cette intersection n'existe pas ou, au contraire, n'est pas unique. Nous montrerons comment le problème a été résolu.

Pour illustrer ces propos, considérons le cas d'une application où l'on cherche à déplacer un objet le long d'une droite paramétrique. En simplifiant quelque peu, la fonction de l'interface (indépendante de l'application) qui s'occupe de gérer la contrainte s'écrit comme

[1] Les systèmes de gestion de fenêtres permettent en général de préciser ce qu'est un mouvement significatif. Par exemple, sous environnement X11, il est possible de ne recevoir des évènements indiquant un déplacement de la souris que lorsqu'elle a bougé de plus d'un certain nombre de pixels

suit:

```
translationlelongdunedroite( p, v, f)
        Point p;
        Vecteur v;
        Fonction f;
{
        double t_prec = 0
        double t; /* t paramétrise la droite représentant la contrainte */
        char fin = FAUX;

        répéter {
                lire un évènement
                si l'évènement est un déplacement de la souris {
                        récupérer les positions x et y de la souris
                        calculer l'équation de la droite œil-curseur à partir de x et y
                        t = intersection entre cette droite et la droite P+tV
                        Δt = t − tprec
                        appeler la fonction f
                }
                si l'évènement est le relachement du bouton {
                        fin = VRAI
                }
        } tant que fin = FAUX;
}
```

Quant à elle, la fonction f se schématise de la façon suivante:

```
f()
{
        Vecteur v_trans

        v_trans = Δt * V
        effacer l'objet manipulé
        appliquer la translation de vecteur v_trans à cet objet
        afficher l'objet translaté
}
```

2.2.1 Equation de la droite *œil-curseur*

Le principe de l'interaction repose sur l'utilisation de la droite passant par la souris et orthogonale au plan de l'écran. De manière à déterminer les intersections entre cette

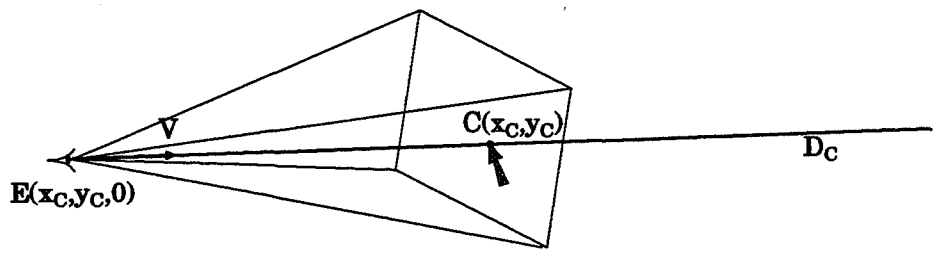


FIG. 2.3 Calcul de la droite *œil-curseur*

droite et les objets de la scène, il nous faut déterminer l'équation de cette droite dans le repère de la scène. Pour ceci, nous disposons des coordonnées du curseur (x_c, y_c) dans l'écran et de la matrice de projection (M) du repère du monde dans le repère de l'écran.

Dans le repère de l'écran, un point P de la droite D a pour coordonnées (x_c, y_c, z) où x_c et y_c représentent toujours la position de la souris alors que z peut prendre toute valeur positive (figure 2.3). Dans le repère du monde, les coordonnées de P sont obtenues à l'aide de M^{-1}, l'inverse de la matrice[2] M:

$$P = \left(\frac{P_x}{P_w} \quad \frac{P_y}{P_w} \quad \frac{P_z}{P_w} \right)$$

avec

$$\begin{aligned} P_x &= M_{11}^{-1}x_c + M_{12}^{-1}y_c + M_{13}^{-1}z + M_{14}^{-1} \\ P_y &= M_{21}^{-1}x_c + M_{22}^{-1}y_c + M_{23}^{-1}z + M_{24}^{-1} \\ P_z &= M_{31}^{-1}x_c + M_{32}^{-1}y_c + M_{33}^{-1}z + M_{34}^{-1} \\ P_w &= M_{41}^{-1}x_c + M_{42}^{-1}y_c + M_{43}^{-1}z + M_{44}^{-1} \end{aligned}$$

L'œil étant caractérisé par une profondeur nulle dans le repère normalisé de l'écran, ses coordonnées sont $(x_c, y_c, 0)$. Dans le repère du monde, ses coordonnées sont données par:

$$E = \left(\frac{E_x}{E_w} \quad \frac{E_y}{E_w} \quad \frac{E_z}{E_w} \right)$$

avec

$$\begin{aligned} E_x &= M_{11}^{-1}x_c + M_{12}^{-1}y + M_{14}^{-1} \\ E_y &= M_{21}^{-1}x_c + M_{22}^{-1}y + M_{24}^{-1} \\ E_z &= M_{31}^{-1}x_c + M_{32}^{-1}y + M_{34}^{-1} \\ E_w &= M_{41}^{-1}x_c + M_{42}^{-1}y + M_{44}^{-1} \end{aligned}$$

Un vecteur directeur de la droite D est par exemple le vecteur $V = EM$ dont les coordonnées dans le repère du monde sont:

$$E = \left(\frac{M_x}{M_w} - \frac{E_x}{E_w} \quad \frac{M_y}{M_w} - \frac{E_y}{E_w} \quad \frac{M_z}{M_w} - \frac{E_z}{E_w} \right)$$

[2] M^{-1} ne nécessite d'être calculée qu'à chaque modification du point de vue, donc de M.

En effectuant les calculs, de nombreuses simplifications surviennent; par exemple en ce qui concerne l'axe des x, on a:

$$\begin{aligned} V_x &= \frac{P_x E_w - E_x P_w}{P_w E_w} \\ &= \frac{M_{13}^{-1} z E_w - M_{43}^{-1} E_x}{P_w E_w} \\ &= t M_{13}^{-1} E_w - M_{43}^{-1} E_x \end{aligned}$$

De même,

$$\begin{aligned} V_y &= t M_{23}^{-1} E_w - M_{43}^{-1} E_y \\ V_z &= t M_{33}^{-1} E_w - M_{43}^{-1} E_z \end{aligned}$$

où le paramètre t est donné par:

$$t = \frac{z}{P_w E_w}$$

Dans le repère du monde, l'équation de la droite passant par le curseur est définie sous forme paramétrique par:

$$E + tV$$

2.3 Les contraintes de mouvement

Les contraintes permettent de limiter les mouvements des objets manipulés à des sous-espaces de $\mathcal{R}^3$. Tout sous-espace de dimension une ou deux peut être pris en compte, pourvu que l'on puisse en déterminer la ou les intersections avec une droite. Une caractéristique géométrique qui revient dans chaque contrainte est le point P. Il représente le point de l'objet sur lequel a cliqué l'utilisateur lors du choix de l'objet à déplacer. En fait, chaque contrainte doit contenir ce point de manière à éviter un brusque déplacement de l'objet lors du premier déplacement de la souris. En particulier, on recense:

- les contraintes ne disposant que d'un degré de liberté représenté par un paramètre t qui sont:
  - La droite caractérisée par un point P et un vecteur V. Cette contrainte permet de réaliser des translations le long de la droite $P + tV$. On les appelle translations-1D.
  - Le cercle caractérisé par un centre C, le point P situé sur la circonférence et un axe V orthogonal au plan du cercle. C ne se situe pas forcément dans le plan passant par P et orthogonal à V Le vrai centre se calcule aisément. Ici, t paramétrise l'angle de rotation entre la position initiale P et les positions projetées successivement sur le cercle. Cette contrainte permet d'effectuer des rotations autour d'un axe, appelées rotation-1D.

  - Une courbe paramétrique dont les caractéristiques géométriques sont des points de contrôle. Si l'on utilise des splines cubiques uniformes, t est contenu dans $[0,1]$.

- Les contraintes dotées de deux degrés de liberté représentés par les paramètres u et v. Ce sont:

  - Le plan passant par P le point initial et doté de deux vecteurs directeurs X et Y. Cette contrainte permet de réaliser des translations dans le plan $P + uX + vY$, appelées translations-2D.
  - La sphère caractérisée par un centre C, un point de sa surface P et deux vecteurs X et Y qui représentent les axes autour desquels peuvent se faire des rotations bi-dimensionnelles, appelées rotations-2D.
  - La surface paramétrique définie par un maillage de points de contrôle.

Nous n'avons pas trouvé d'application, dans le cadre de nos travaux sur les interpolations d'orientation, aux espaces de contraintes définies par des courbes ou surfaces paramétriques. Cet interface pourrait cependant s'avérer utile dans le cadre d'un modeleur manipulant de telles primitives. On peut par exemple envisager de déplacer un objet sur une surface paramétrique.

2.4 Intersections généralisées

L'évaluation des valeurs des paramètres t ou u et v se fait en intersectant l'espace de la contrainte de mouvement avec la droite *œil-curseur*. Contrairement à ce qui est fait en tracé de rayon – où le résultat de l'intersection fournit une réponse booléenne suivant que la primitive est intersectée ou non – il faut ici assurer qu'une intersection fournit toujours un résultat, même si ce n'est pas l'intersection exacte. Pour cela, nous introduisons la notion d'intersection généralisée, notée $\cap^*$, entre un espace E (le sous-espace des contraintes) et une droite D (il s'agit en l'occurrence de la droite *œil-curseur*). Elle se définit de la manière suivante:

- $E \cap D$ admet une solution unique : $E \cap^* D = E \cap D$. C'est principalemet le cas quand E est un plan (figure 2.4).

- $E \cap D$ n'admet pas de solution. Ce cas se produit fréquemment quand E est un cercle, une droite, une courbe paramétrique ou une sphère. On détermine alors un point de E qui fera office d'inverse généralisée (figure 2.5). Quand E est une droite, $E \cap^* D$ est alors le point sur E le plus proche de D. Dans les autres cas, on utilise des constructions géométriques. Les méthodes sont détaillées ci-dessous.

- Quand E est une sphère (figure 2.6.a) ou une surface paramétrique, $E \cap D$ admet plusieurs solutions. Les calculs d'intersection sont les mêmes que ceux utilisés dans l'algorithme du tracé de rayons. Parmi les solutions, on retient l'intersection la plus

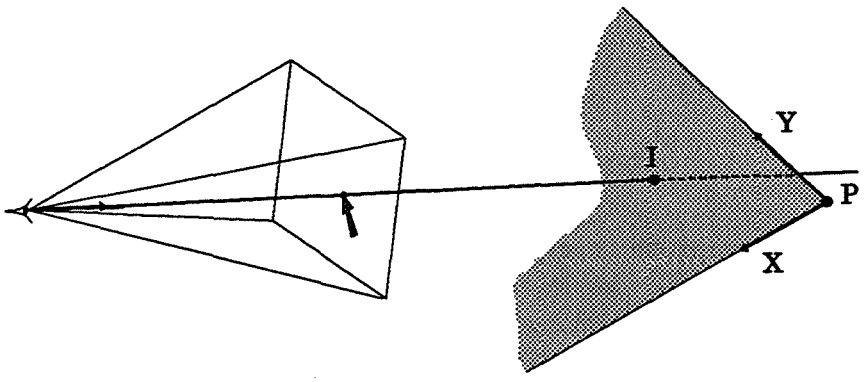


FIG. 2.4 Intersection unique

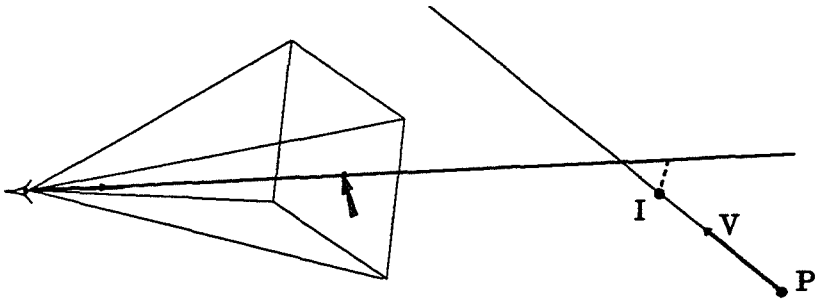


FIG. 2.5 Intersection vide

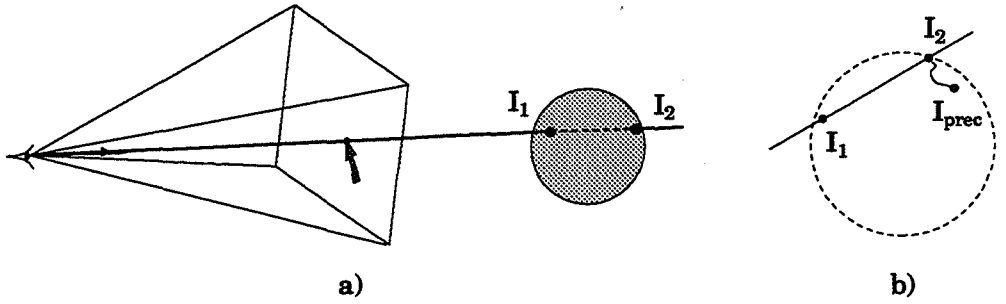


FIG. 2.6 Intersections multiples

proche de l'intersection trouvée à l'étape précédente (juste avant le dernier mouvement de la souris). En d'autres termes, c'est celle qui introduit le moins de variations dans les paramètres qui est retenue. Sur la figure (2.6.b), l'intersection précédente est notée I_{prec} et c'est l'intersection I_2 qui est retenue.

Remarque

Dans les cas où il n'y a pas d'intersection réelle, l'intersection généralisée détermine un point ne se situant plus sur la droite *œil-curseur*. A l'écran, il est possible que l'objet se trouve déplacé en un point éloigné de la position de la souris. Pour éviter de déconcerter l'utilisateur, le curseur est déplacé arbitrairement sur la projection de l'intersection généralisée à l'écran.

On détaille maintenant la résolution des intersections généralisées dans le cas de la droite, du cercle et de la sphère.

2.4.1 Intersection généralisée droite-droite

Il y a rarement intersection entre deux droites dans $\mathcal{R}^3$. Soit $P_c + t_c V_c$ la droite passant par les coordonnées de la souris, orthogonale au plan de l'écran, et $P + tV$ l'équation de la droite de contrainte. On choisit de retrouver le point de la droite de contrainte le plus proche des deux droites. Si D désigne la distance entre ces deux droites, nous avons :

$$\begin{aligned} D^2 &= (P_c + t_c V_c - (P + tV))^2 \\ &= (P_c + t_c V_c)^2 + (P + tV)^2 - 2(P_c + t_c V_c)(P + tV) \\ &= P_c^2 + t_c^2 V_c^2 + 2P_c t_c V_c + P^2 + t^2 V^2 + 2PtV - 2P_c P \\ &\quad -2P_c tV - 2t_c V_c P - 2t_c t V_c V \end{aligned}$$

La distance est minimale quand les dérivées partielles par rapport aux paramètres t_c et t s'annulent, c'est à dire:

$$\frac{\partial D^2}{\partial t_c} = 2t_c V_c^2 + 2P_c V_c - 2V_c P - 2tV_c V = 0$$

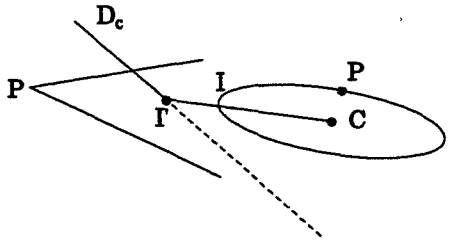


FIG. 2.7 Intersection droite-cercle

$$\frac{\partial D^2}{\partial t} = 2V^2t - 2V_cV + 2PV - 2t_cV_cV = 0$$

Cela revient à résoudre le système de deux équations à deux inconnues suivant:

$$\begin{cases} V_c^2t_c - V_cVt + V_c(P - P_c) = 0 \\ -V_cVt_c + V^2t + V(P - P_c) = 0 \end{cases}$$

qui admet une solution tant que V et V_c ne sont pas colinéaires.

2.4.2 Intersection généralisée droite-cercle

Le cercle est défini par un axe V, un centre C et un point P de sa circonférence. On commence par intersecter la droite *œil-curseur* D_c avec le plan Π du cercle. C'est le plan passant par P et orthogonal à V (figure 2.7). Cela nous fournit un point I' qui est alors projeté sur le cercle. L'intersection entre le cercle et I' est donnée par:

$$I = C + CI'\frac{|CI'|}{|CP|}$$

2.4.3 Intersection généralisée droite-sphère

On détecte l'intersection entre la droite *œil-curseur* et la sphère avec la méthode classique du tracé de rayon. Si aucune intersection n'est détectée, on cherche le point sur D_c le plus proche du centre C de la sphère par une méthode analogue à celle du § 2.4.1. La distance entre les deux est donnée par:

$$D^2 = (P_c + t_cV_c - C)^2$$

La distance est minimale quand:

$$\frac{\partial D^2}{\partial t_c} = 2V^2t_c + 2V(P - C) = 0$$

c'est à dire

$$t_c = \frac{V(P-C)}{V^2}$$

Cette valeur du paramètre fournit le point de D_c le plus proche de la sphère. Il est alors facile de trouver l'intersection entre ce point et la sphère. Elle représente l'intersection généralisée entre la droite et la sphère.

2.5 Interaction sur un objet articulé

Dans ce paragraphe, nous proposons un algorithme de cinématique inverse permettant d'effectuer le placement interactif d'un solide rigide articulé. Construit comme une sur-couche du système d'interaction exposé ci-dessus, il permet de définir rapidement les positions et orientations-clés d'un objet complexe.

2.5.1 Introduction

L'animation de solides rigides articulés, tels que des personnages humains, est la source de nombreux problèmes [BOK80]. Nous n'aborderons pas ici le problème du contrôle du mouvement mais les moyens que l'on peut offrir à un animateur pour définir les positions du corps articulé. On distingue principalement deux types d'opérations permettant de modifier l'allure d'un solide articulé. La première, appelée cinématique directe consiste à déterminer les positions de ses différents éléments en fonction des variables articulaires. A l'opposé, des objectifs tels qu'avancer un pied ou saisir un objet doivent être résolus par les méthodes de la cinématique inverse. Il s'agit de déterminer les variables articulaires à partir du déplacement relativement à un repère de référence d'un ou de plusieurs éléments du solide articulé. Les solutions aux problèmes de la cinématique inverse sont nombreux mais souvent complexes en raison de la redondance des nombreux degrés de liberté [BKK+86]. Si le problème de cinématique inverse peut être résolu analytiquement dans certains cas, la résolution doit en général faire appel à des méthodes numériques (utilisation de la matrice jacobienne, méthodes d'optimisation) résumées dans [KB86].

Cette partie est essentiellement consacrée à la présentation d'un nouvel algorithme de cinématique inverse. Sa rapidité autorise son utilisation dans une interface de haut niveau permettant d'éditer les positions clés d'un solide articulé. Pour atteindre cet objectif, une approche différente de celles couramment employées est proposée. En effet, notre algorithme ne se sert que des positions des différents éléments constituant le solide articulé.

2.5.2 Modélisation du solide articulé

Classiquement, on modélise un solide articulé sous la forme d'un arbre d'articulations. Dans notre application, les articulations ne sont reliées entre elles que par des liaisons

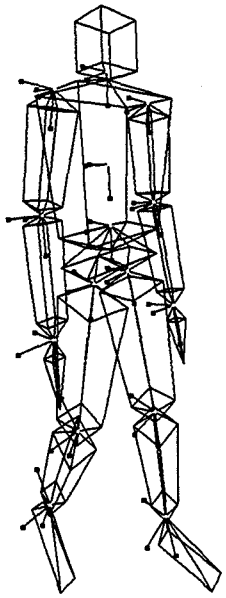


FIG. 2.8 Un solide articulé : chaque articulation dispose de son repère local

rotoïdes (trois degrés de liberté de rotation). Une articulation est géométriquement définie par une position et une orientation à l'aide des deux paramètres suivants :

- Une orientation relative à l'orientation du père, représentée par le quaternion q_{local}.
- Une position définie dans le repère local du père, représentée par le vecteur p_{local}. Ce vecteur est toujours constant.

A partir de ces données, les positions et orientations dans le repère du monde sont calculées par un parcours préfixé de l'arbre des articulations.

Considérons un nœud n :
si n est la racine
$q_{ref}(n) = q_{local}(n)$
$p_{ref}(n) = p_{local}(n)$
sinon
$q_{ref}(n) = q_{ref}(\text{père}(n)).q_{local}(n)$
$p_{ref}(n) = p_{ref}(\text{père}(n)) + p_{local}(n)$

Enfin, chaque élément est dotée d'une enveloppe géométrique définie dans son repère local (figure 2.8).

2.5.3 Contrôle interactif

Ici, les repères locaux vont nous servir à spécifier les contraintes de mouvement requises par l'interface 3D. Une représentation de ces repères est projetée sur l'écran ce qui permet à l'utilisateur de sélectionner ce que nous appelerons un axe principal. Le repère local contenant l'axe principal est appelé repère principal. Ensuite, il peut agir directement sur les points de l'enveloppe du solide. Le déplacement du point sélectionné se fait, en fonction des contraintes de mouvement choisies, soit en rotation, soit en translation par rapport à l'axe principal.

Il faut ensuite répercuter les mouvements de ce point sur l'ensemble du solide articulé. Par exemple, si l'utilisateur déplace une des mains, il faut que le reste du corps suive le mouvement.

Deux solutions sont proposées à l'utilisateur. La première consiste à déplacer un point de l'enveloppe par translation ou rotation. On fait ensuite appel à l'algorithme de cinématique inverse présenté au paragraphe suivant pour déterminer les nouvelles variables articulaires (figure 2.9.a).

Dans le deuxième cas, le traitement dépend du type de mouvement effectué. La translation (en une ou deux dimensions) d'un point du corps articulé entraîne une translation équivalente de l'ensemble de l'objet (figure 2.9.b). Si le mouvement est une rotation, on distingue les cas où le point déplacé est, ou non, un descendant de l'articulation possédant l'axe principal :

- Si c'est un descendant, on applique au sous-arbre de ce point une rotation équivalente à la rotation du point déplacé (figure 2.9.c). Il suffit pour cela de modifier l'orientation du repère principal.
- Dans le cas contraire, les descendants du repère principal demeurent inchangés, mais on tourne tous les ascendants. Cela revient à modifier l'orientation et la position de la racine ainsi que l'orientation locale du repère principal (figure 2.9.d).

2.5.4 Cinématique inverse sur un bras articulé

La chaîne articulée est représentée par n segments reliant entre eux les points $\{P_1, \ldots, P_{n+1}\}$ (figure 2.10.a).

Le problème est de déterminer les nouvelles coordonnées de ces $n+1$ points si l'on déplace l'un d'entre eux. Contrairement aux méthodes plus classiques, on ne cherche pas à déterminer les angles entre les différents segments.

Le principe de l'algorithme consiste à autoriser les différents éléments du bras à s'étirer ou se rétrécir. Par exemple, le déplacement du point P_{n+1} en P'_{n+1} augmente la longueur du segment terminal. Il s'agit ensuite, par une solution itérative, de faire revenir chaque segment à sa longueur naturelle de manière à ce que la chaîne articulée retrouve sa cohérence initiale[3] (figure 2.10.b).

[3]Dans le même ordre d'idée, noter l'approche présentée dans [vO91]. Chaque élément du bras est déplacé

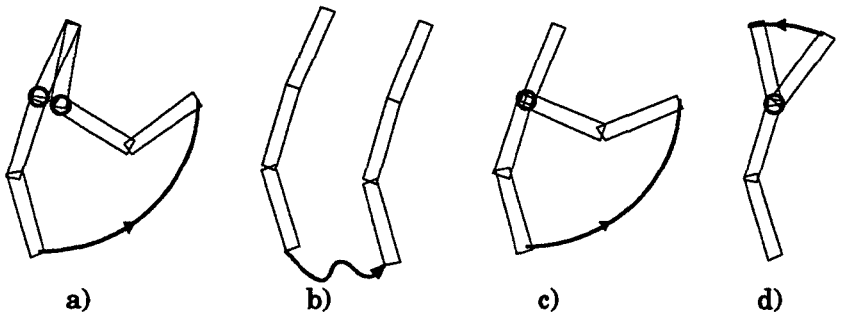


FIG. 2.9 L'axe principal (représenté par le cercle) est orthogonal au plan de la figure, la racine du solide est en haut. a) Rotation avec résolution par cinématique inverse. b) Translation en deux dimensions dans le plan de la figure . c) Rotation des descendants. d) Rotation de l'ascendant à l'axe principal.

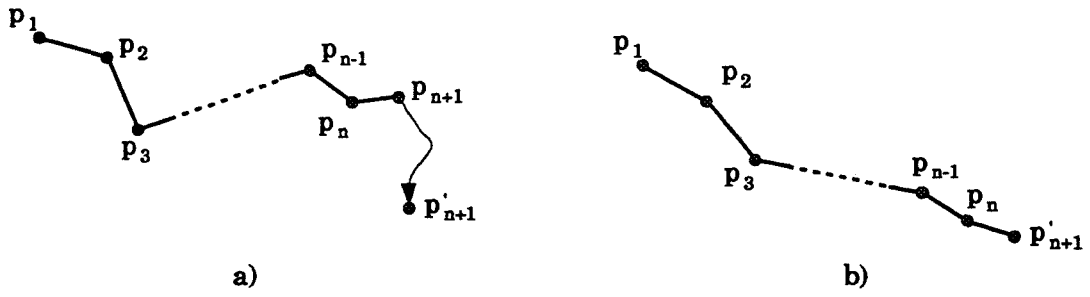


FIG. 2.10 Représentation d'une chaîne articulée

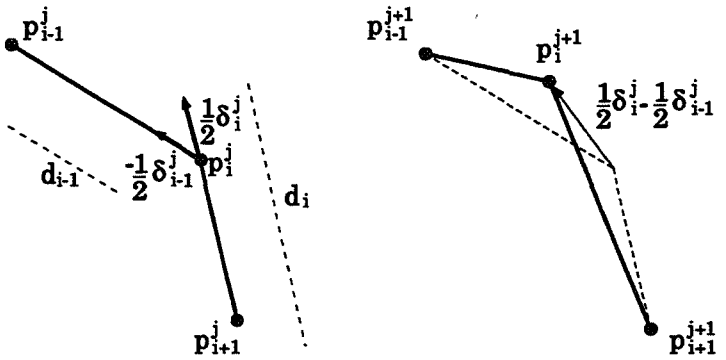


FIG. 2.11 Passage de l'étape j à l'étape $j+1$ sur l'articulation i

Nous utiliserons les notations suivantes :

- p_i^j : position de l'articulation i à l'étape j.
- d_i : distance initiale entre les points P_i et P_{i+1}, appellée longueur naturelle.
- l_i^j : longueur du segment i à l'étape j.
- Δ_i^j : écart entre la longueur naturelle et la longueur courante du segment i à l'étape j, i.e. $\Delta_i^j = l_i^j - d_i$.

Pour rétablir les longueurs naturelles, on applique aux extrémités de chaque segment, une translation $\frac{1}{K}\delta_i^j$[4] ($K \geq 2$) proportionnelle à l'écart entre sa longueur actuelle et sa longueur naturelle. Dans le cas de la figure, seuls les deux derniers segments du bras articulé seront déplacés. Au fur et à mesure des itérations successives, les déplacements seront propagés vers la base de la chaîne.

Chaque point du bras subit une translation qui est la somme des déplacements des deux segments auxquels il appartient (figure 2.11). L'algorithme s'arrête quand les longueurs des segments sont suffisamment proches de leurs longueurs naturelles. Une variable notée ϵ contrôle cette notion de proximité. Pour accélérer la résolution, on utilise une valeur assez grande tant que l'utilisateur déplace un élément du bras. A la fin de l'interaction (dès que le bouton de la souris est relâché), on applique la cinématique inverse une dernière fois avec une valeur très petite.

---

indépendamment des autres en utilisant des lois de la dynamique. Des itérations successives permettent de recoller les morceaux grâce à des contraintes géométriques.

[4]K intervient sur la vitesse de convergence de l'algorithme. On verra plus loin que la convergence optimale est obtenue pour $K = 2$.

L'algorithme est résumé par les instructions suivantes:

j = 0
répéter {
$\delta_i^j = \frac{|p_i^j p_{i+1}^j| - d_i}{|p_i^j p_{i+1}^j|} p_i^j p_{i+1}^j \quad 2 \leq i \leq j$
$\delta_1^j = 0; \delta_{n+1}^j = 0$
$p_i^{j+1} = p_i^j + \frac{\delta_i^j}{K} - \frac{\delta_{i-1}^j}{K} \quad 2 \leq i \leq j$
$j = j + 1$
} **tant que** $\sum_{i=1}^{n} ||p_i^{j+1} p_{i+1}^{j+1}| - d_i| > \epsilon$

Remarque

Les points P_1 et P_{n+1} ne sont pas affectés par les translations. De la même manière, on peut fixer si besoin est, d'autres points de la chaîne.

Dans le paragraphe qui suit, nous montrons qu'à chaque itération, le bras articulé se rapproche de sa configuration naturelle, c'est à dire que l'écart entre la longueur d'un segment et sa longueur naturelle décroît à chaque itération. Cette démonstration n'assure pas la convergence théorique de l'algorithme. En pratique, celle-ci est néanmoins vérifiée.

On pourra trouver dans [GG92], une démonstration généralisée au cas des graphes.

Démonstration de la décroissance. Considérons le segment (P_i, P_{i+1}) à l'étape j.

A l'étape suivante, on aura:

$$p_i^{j+1} = p_i^j + \frac{\delta_i^j}{K} - \frac{\delta_{i-1}^j}{K} \tag{2.1}$$

$$p_{i+1}^{j+1} = p_{i+1}^j + \frac{\delta_{i+1}^j}{K} - \frac{\delta_i^j}{K} \tag{2.2}$$

A partir de ces deux équations (2.1) et (2.2) et de la figure (2.12), on voit que l'on peut encadrer la longueur du segment à l'étape $j+1$ de la façon suivante:

$$l_i^j - \frac{2}{K}\Delta_i^j - \frac{\Delta_{i-1}^j}{K} - \frac{\Delta_{i+1}^j}{K} \leq l_i^{j+1} \leq l_i^j - \frac{2}{K}\Delta_i^j + \frac{\Delta_{i-1}^j}{K} + \frac{\Delta_{i+1}^j}{K}$$

$$\frac{1}{K}\left(l_i^j(K-2) + 2d_i - \Delta_{i-1}^j - \Delta_{i+1}^j\right) \leq l_i^{j+1} \leq \frac{1}{K}\left(l_i^j(K-2) + 2d_i + \Delta_{i-1}^j + \Delta_{i+1}^j\right)$$

$$\frac{1}{K}\left(\left(l_i^j - d_i\right)(K-2) - \Delta_{i-1}^j - \Delta_{i+1}^j\right) \leq l_i^{j+1} - d_i \leq \frac{1}{K}\left(\left(l_i^j - d_i\right)(K-2) + \Delta_{i-1}^j + \Delta_{i+1}^j\right)$$

et comme $\Delta_i^{j+1} = l_i^{j+1} - d_i$:

$$\frac{K-2}{K}\Delta_i^j - \frac{\Delta_{i-1}^j}{K} - \frac{\Delta_{i+1}^j}{K} \leq \Delta_i^{j+1} \leq \frac{K-2}{K}\Delta_i^j + \frac{\Delta_{i-1}^j}{K} + \frac{\Delta_{i+1}^j}{K} \tag{2.3}$$

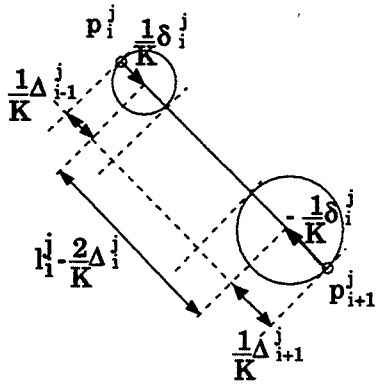


FIG. 2.12 Variation de la longueur d'un segment de bras

En posant $A = \frac{K-2}{K}\Delta_i^j$ et $B = \frac{\Delta_{i-1}^j}{K} + \frac{\Delta_{i+1}^j}{K}$, on réécrit (2.3) sous la forme condensée:

$$A - B \leq \Delta_i^{j+1} \leq A + B$$

Discutons suivant le signe de A:

- si $A < 0$

$$\begin{aligned} A - B \leq \quad & \Delta_i^{j+1} \quad \leq -A + B \\ -(B - A) \leq \quad & \Delta_i^{j+1} \quad \leq B - A \\ |\Delta_i^{j+1}| \quad & \leq |B - A| \end{aligned}$$

- sinon

$$\begin{aligned} -A - B \quad & \leq \Delta_i^{j+1} \leq \quad A + B \\ -(A + B) \leq \quad & \Delta_i^{j+1} \quad \leq A + B \\ |\Delta_i^{j+1}| \quad & \leq |A + B| \end{aligned}$$

d'où

$$\begin{aligned} |\Delta_i^{j+1}| \quad & \leq \quad \max(|A + B|, |A - B|) \\ |\Delta_i^{j+1}| \quad & \leq \quad \frac{K-2}{K}|\Delta_i^j| + |\frac{\Delta_{i-1}^j}{K}| + |\frac{\Delta_{i+1}^j}{K}| \end{aligned} \tag{2.4}$$

Il peut arriver que l'algorithme oscille indéfiniment si les trois points P_{i-1}, P_i et P_{i+1} sont alignés. Dans le cas contraire, on peut remplacer l'inégalité par une inégalité stricte. D'autre part, on voit ici que la convergence optimale est obtenue quand $K = 2$.

Examinons maintenant l'évolution de la somme des variations de longueur au cours de l'algorithme. En sommant l'inéquation (2.4) sur tous les segments, on a:

$$\sum_{i=1}^{n} |\Delta_i^{j+1}| \leq \frac{1}{K}\left((K-2)\sum_{i=1}^{n}|\Delta_i^j| + \sum_{i=1}^{n}|\Delta_{i-1}^j| + \sum_{i=1}^{n}|\Delta_{i+1}^j|\right)$$

$$\sum_{i=1}^{n}|\Delta_i^{j+1}| \leq \frac{1}{K}\left(2\sum_{i=2}^{n+1}|\Delta_i^j| + |\Delta_0^j| + |\Delta_1^j| + |\Delta_n^j| + |\Delta_{n+1}^j|\right) + (K-2)\left(\sum_{i=2}^{n-1}|\Delta_i^j| + |\Delta_1^j| + |\Delta_n^j|\right)$$

$$\sum_{i=1}^{n}\Delta_i^{j+1} \leq \frac{1}{K}\left(\sum_{i=2}^{n-1}|\Delta_i^j| + (K-1)\left(|\Delta_1^j| + |\Delta_n^j|\right) + |\Delta_0^j| + |\Delta_{n+1}^j|\right)$$

or $\Delta_0^j = \Delta_{n+1}^j = 0$, donc

$$\sum_{i=1}^{n}|\Delta_i^{j+1}| \leq \frac{1}{K}\left(K\sum_{i=2}^{n+1}|\Delta_i^j| - \left(|\Delta_1^j| + |\Delta_n^j|\right)\right)$$

$$\sum_{i=1}^{n}|\Delta_i^{j+1}| \leq \sum_{i=1}^{n}|\Delta_i^j| - \frac{1}{K}\left(|\Delta_1^j| + |\Delta_n^j|\right)$$

et finalement

$$\sum_{i=1}^{n}|\Delta_i^{j+1}| \leq \sum_{i=1}^{n}|\Delta_i^j|$$

A nouveau, si au moins trois des points constituant le bras ne sont pas alignés (i.e. si le bras ne décrit pas une droite), on peut mettre une inégalité stricte :

$$\sum_{i=1}^{n}|\Delta_i^{j+1}| < \sum_{i=1}^{n}|\Delta_i^j|$$

ce qui achève la démonstration. En pratique, à chaque itération les écarts Δ_i sont quasiment divisés par deux.

2.5.5 Cinématique inverse sur un arbre

Le principe de l'algorithme est identique au précédent, excepté que, de par la structure du modèle, celui-ci est maintenant récursif. A chaque nœud est appliqué un déplacement proportionnel à celui que subissent son père et chacun de ses fils (le même principe est utilisé dans [GG92] pour maintenir des contraintes). La première étape de l'algorithme

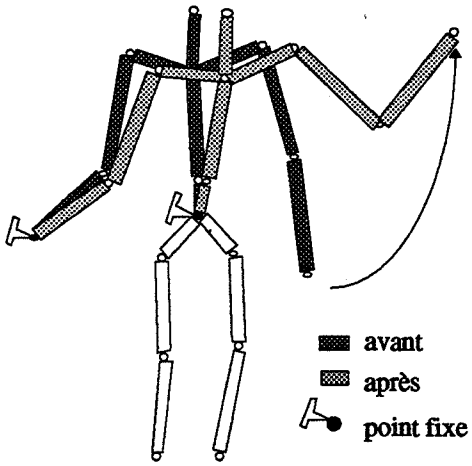


FIG. 2.13 Cinématique inverse contrainte

calcule les déplacements de chaque articulation, la seconde effectue les translations.

CinematiqueInverse(a) {
pour chaque $f \in$ fils(a) **faire** {
$\delta_f^j = \frac{|p_a^j p_f^j| - d_f}{|p_a^j p_f^j|} p_a^j p_f^j$
}
appeler l'algorithme pour chaque fils de a
pour chaque $f \in$ fils(a) **faire** {
$p_a^{j+1} = p_a^j + \delta_a^j - \frac{\sum_{f \in \text{fils}(a)} \delta_f^j}{1+\text{nombre de fils}(a)}$
}
}

Ici, d_f désigne la longueur naturelle entre l'articulation f et son père.

On appelle cet algorithme, avec comme argument la racine du solide, le nombre de fois nécessaires à l'obtention de la convergence.

La particularité de la méthode fait que l'on peut aisément contraindre certains points du squelette à rester fixes (on se rapproche là des contraintes de position présentées dans [BMW87]). Il suffit de ne pas déplacer chaque articulation contrainte. On voit un exemple sur la figure (2.13) où la main gauche et le bassin restent en place, alors que la main droite est déplacée.

Plus généralement, il est tout à fait possible d'imposer une trajectoire prédéfinie à tout élément de l'arbre. De même, cette méthode qui est utilisée sur une topologie d'arbre peut aisément être appliquée à des solides ayant une topologie de graphe. Il suffit de calculer les déplacements de chaque nœud du graphe avant de les appliquer à chacune de ses arêtes.

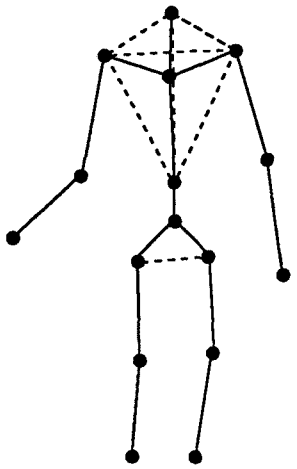


FIG. 2.14 Liaisons supplémentaires maintenant l'intégrité du corps articulé

2.5.6 Maintien de la forme du solide

L'algorithme ci-dessus ne tient pas compte des contraintes supplémentaires éventuelles pouvant exister entre chacun des fils d'un même solide. Aussi, aux liens père-fils utilisés dans l'algorithme ci-dessus, ajoute-t-on un ensemble de liaisons fictives reliant les frères entre eux. De cette manière, le principe de la méthode est conservé tout en maintenant l'intégrité du solide articulé. A chaque articulation disposant de n fils contraints ($n > 1$), on introduit $\frac{n(n+1)}{2}$ liaisons supplémentaires (figure 2.14).

Cette obligation de rajouter des contraintes provient de la spécificité de l'algorithme de cinématique inverse à ne traiter que des points. Traiter le solide articulé comme un ensemble d'objets rigides reliés par des liaisons nécessiterait de modifier le principe de l'algorithme. En particulier, en plus des seules translations qui servent à rétablir les contraintes, il faudrait utiliser des rotations et donc tenir compte des orientations des diverses parties de l'objet articulé. C'est ce que fait Gascuel dans [GG92].

2.5.7 Rétablissement des orientations

L'algorithme exposé ci-dessus ne modifie que les positions des articulations du solide. Quand la solution est trouvée, il faut donc remettre à jour les orientations de chaque lien à partir des nouvelles positions calculées. Considérons le cas d'un nœud n entre l'état 0 – avant d'appliquer l'algorithme – et l'état 1 – après résolution de la cinématique inverse – (figure 2.15).

$$\begin{aligned} v^0 &= p^0_{ref}(\text{fils(n)}) - p^0_{ref}(\text{n}) \\ v^1 &= p^1_{ref}(\text{fils(n)}) - p^1_{ref}(\text{n}) \end{aligned}$$

Si $v^0 \neq v^1$, l'orientation du père a changé et il faut la calculer de la façon suivante. On

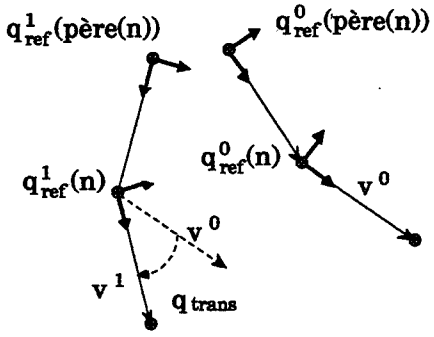


FIG. 2.15 Rétablissement des orientations

commence par déterminer le quaternion q_{trans} qui transforme v^0 en v^1 et on en déduit la nouvelle orientation du père dans le repère du monde :

$$q^1_{ref}(\mathrm{n}) = q_{trans} q_{ref}(\mathrm{n})$$

puis son orientation par rapport l'orientation de son père – qui, elle, a été déterminée à l'étape précédente :

$$q^1_{local}((\mathrm{n}) = \left(q^1_{ref}(\text{père}(\mathrm{n}))\right)^{-1} q^1_{ref}(\mathrm{n})$$

q_{trans} qui permet de passer de l'orientation définie par v^0 à celle définie par v^1 doit vérifier :

$$q_{trans}[0, v^0]q^{-1}_{trans} = [0, v^1]$$

Cette équation admet une infinité de solutions, aussi, choisit-on arbitrairement comme axe de rotation un vecteur orthogonal à v^0 et v^1. L'angle de rotation θ est donné par :

$$\theta = \cos^{-1}\left(\frac{v^0}{\|v^0\|} \cdot \frac{v^1}{\|v^1\|}\right)$$

d'où

$$q_{trans} = \left[\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\left(\frac{v^0}{\|v^0\|} \wedge \frac{v^1}{\|v^1\|}\right)\right]$$

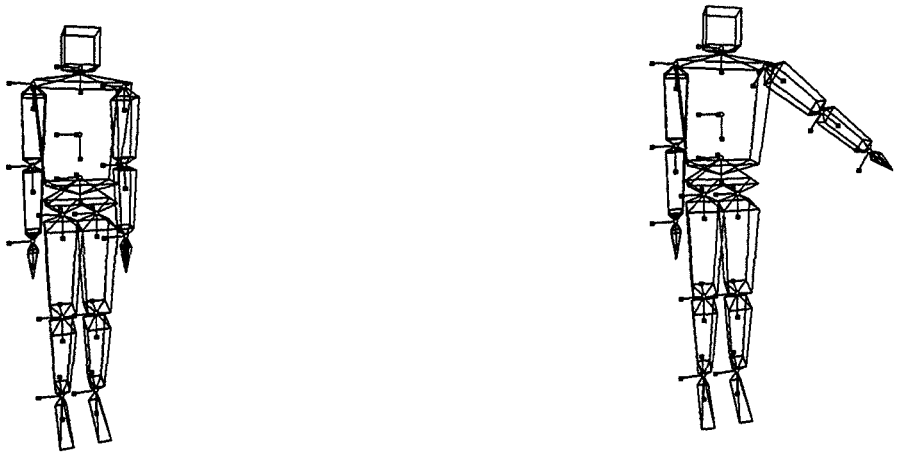
2.5.8 Exemples

La figure 2.16 montre trois transformations d'un solide articulé obtenues en appliquant les techniques présentées :

a) Solide articulé dans sa position initiale.

b) Rotation du bras gauche autour d'un axe de l'épaule; le bras tourne en suivant le déplacement de la main.

c) Translation en deux dimensions de la main gauche avec résolution par cinématique inverse; les pieds et la main droite sont fixes.



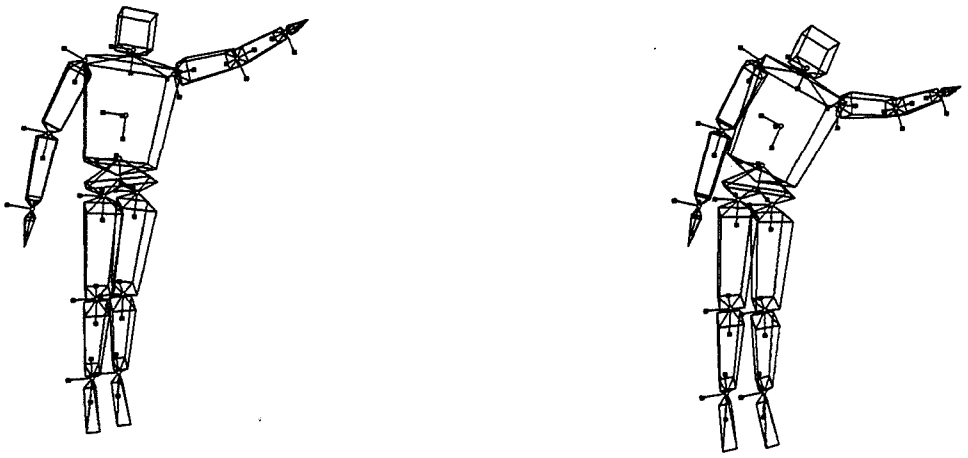
a) b)

d) Translation en deux dimensions de la main gauche avec résolution par cinématique inverse sans contraintes de position.

2.6 Conclusion

Dans la première partie de ce chapitre, nous avons présenté un système d'interfaçage entre des applications 3D et une souris. La séparation entre l'application et l'interface est clairement mise en évidence. Cette modularité requiert par contre que l'application gère elle-même la gestion des contraintes de mouvement. En contrepartie, toutes sortes de contraintes peuvent être prises en compte. Enfin, malgré sa souplesse d'emploi, le noyau de cet interface demeure simple. Ses fonctions essentielles sont le calcul de la droite passant par la souris et la résolution des intersections généralisées entre les contraintes et cette droite.

Dans la deuxième partie, nous avons décrit une application utilisant cet interface. Elle permet de manipuler en temps réel un solide articulé. De par sa simplicité, l'algorithme de cinématique inverse présenté est facile à mettre en œuvre. En contrepartie, certaines possibilités offertes par d'autres algorithmes de cinématique inverse sont écartées. Plus précisément, nous ne traitons ni les limites angulaires (butées), ni les liaisons à un ou deux degrés de liberté.



c) d)

FIG. 2.16 Manipulation d'un solide articulé

Chapitre 3

Cinétique du mouvement

3.1 Introduction

Dans les paragraphes précédents, nous ne nous sommes préoccupés que des trajectoires géométriques des objets animés, qu'elles concernent les positions ou les orientations. Cependant, la définition d'une trajectoire ne suffit pas à spécifier un mouvement. Un mouvement peut apparaître spatialement, ou géométriquement, correct mais inapproprié cinétiquement. Un autre paramètre important à maîtriser lors de la création d'animations est la la cinétique[1] du mouvement [Las87].

Typiquement, l'approche présentée dans le premier chapitre utilise la notion de tangente à la trajectoire comme un moyen d'altérer la trajectoire spatiale : augmenter le module d'une tangente augmente l'arrondi de la courbe, le réduire conduit à une trajectoire plus anguleuse.

De manière sous-jacente, ces modifications perturbent aussi les vitesses le long de la courbe. Ils nous semble malgré tout que l'aspect spatial et l'aspect cinétique doivent être traités comme deux phénomènes distincts. Une première raison évidente est que cette séparation offre plus de modularité pendant la création de l'animation. Ensuite, notre modèle d'interpolation, de par la souplesse offerte concernant la définition des courbes spatiales, est peu adapté à la spécification conjointe de la cinétique le long de l'interpolant :

- Les courbes d'Hermite utilisées avec les tangentes des splines cardinales assurent la continuité des dérivées premières. Modifier ces tangentes entraîne la perte de cette continuité. Quand seul le module de la tangente est modifié, on conserve la continuité géométrique au sens de [DB88] (G_1-continuité, la tangente à gauche est colinéaire à la tangente à droite au même point, mais les modules diffèrent). Malgré tout, si l'aspect de la courbe demeure géométriquement acceptable, la cinétique du mouvement ne l'est plus (figure 3.1).

[1] Nous avons choisi d'utiliser le terme de cinétique plutôt que ceux de cinématique ou de dynamique du mouvement. Le mot cinématique fait référence aux trajectoires alors que la cinétique concerne plus spécialement la vitesse du mouvement. Le terme dynamique, couramment usité dans les textes anglais

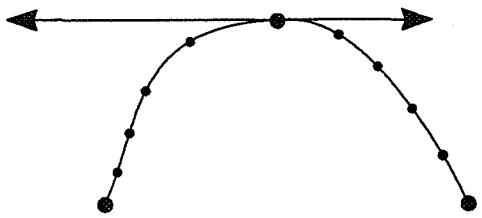


FIG. 3.1 Courbe G_1-continue mais non C_1-continue

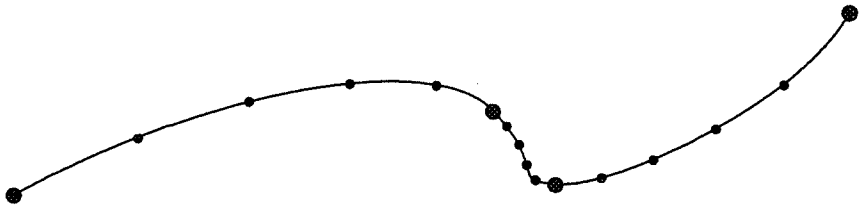


FIG. 3.2 L'espacement entre les points de contrôle influe sur la cinétique du mouvement

- Les courbes d'Hermite sont des splines cubiques uniformes par morceaux. Or, en échantillonnant de manière uniforme le long de la trajectoire globale, on peut engendrer de brusques variations de vitesse en passant d'un point de contrôle au suivant. Le problème vient de ce que l'espacement entre les points de contrôle n'est pas régulier (figure 3.2) et malheureusement les splines cubiques ne tiennent pas compte de la distance entre les points de contrôle. Modifier l'échantillonnage sur chaque tronçon en fonction des distances entre points de contrôle ne résoudrait que partiellement le problème. Parfois, l'utilisation de splines non uniformes ou d'ordre élevé permet, par l'espacement adéquat entre les nœuds, de remédier à cet inconvénient.

- Enfin, la dernière et non la moindre des raisons est que l'utilisateur peut vouloir seul décider de la cinétique de l'objet animé le long de la trajectoire qu'il a préalablement spécifiée.

A ce titre, on distingue essentiellement trois approches, identiques dans le principe, offrant un contrôle sur la cinétique du mouvement. Les trois sont basées sur une reparamétrisation visant à exprimer le paramètre initial u contrôlant la courbe par un autre paramètre plus explicite, par exemple le temps.

(*motion dynamics*), rappelle l'animation dynamique et peut donc prêter à confusion.

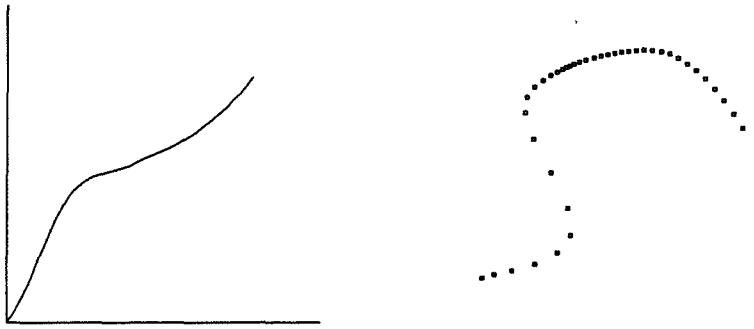


FIG. 3.3 Exemple de trajectoire en dimension deux (Δt = constante). Reparamétrisation à l'aide d'une courbe $u = f(t)$ (a). Le paramètre interpolé est la position (b).

3.2 Travaux antérieurs

3.2.1 Spécification du paramètre en fonction du temps

L'approche de Steketee et Badler [SB85] consiste à définir une courbe additionnelle exprimant directement le paramètre u en fonction du temps. Le mouvement résultant est obtenu par combinaison des deux interpolants (voir figure 3.3). La courbe spatiale est définie par une fonction $P = C(u)$, où P représente l'un des paramètres du corps animé. La reparamétrisation est donnée par une fonction $u = f(t)$. L'évaluation du paramètre à un instant t donné est effectuée par combinaison des deux fonctions : $P = C(f(t))$

Les courbes doivent bien entendu être définies sur des domaines compatibles entre eux. Il suffit de modifier la courbe de reparamétrisation pour changer la cinétique du mouvement, et ceci sans altérer le chemin initial. Par contre, cette méthode dépend encore indirectement du paramètre initial u au travers de la courbe C, ce qui demande une certaine dextérité de la part de l'animateur afin d'en annuler les effets. Pour s'en rendre compte, il suffit de considérer la figure (3.4) où la courbe f est l'identité. La cinétique du mouvement est celle de la spline initiale alors qu'on aurait certainement préféré obtenir un mouvement plus uniforme.

3.2.2 Spécification de l'abscisse curviligne en fonction du temps

La deuxième solution, exposée par exemple dans [YG89], utilise comme paramétrisation une courbe $a = f(t)$ exprimant l'abscisse curviligne le long de la trajectoire en fonction du temps, c'est à dire la distance parcourue le long de la trajectoire en fonction du temps (figure 3.5).

L'évaluation de la valeur du paramètre à un instant t est moins immédiate que dans le cas précédent. Elle se fait de la manière suivante. Il faut d'abord précalculer une fonction supplémentaire $S(u)$ exprimant la longueur parcourue le long de la courbe $C(u)$ en fonction du paramètre u. $S(u)$ est tabulée afin d'assurer de bonnes performances. Ensuite, il suffit de rechercher dans cette table l'abscisse correspondant à l'abcisse à l'instant t, $f(t)$ —

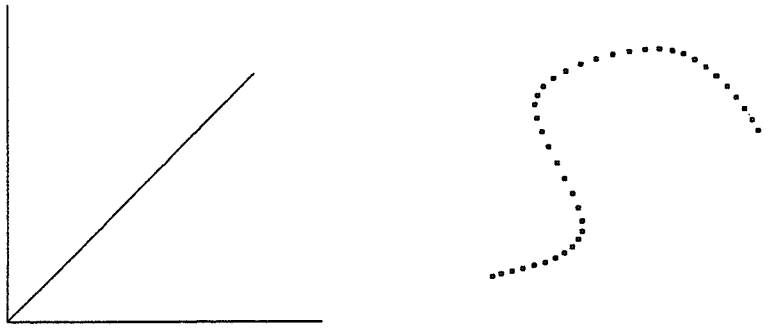


FIG. 3.4 Quand la reparamétrisation est linéaire, on retrouve le mouvement par défaut de la spline.

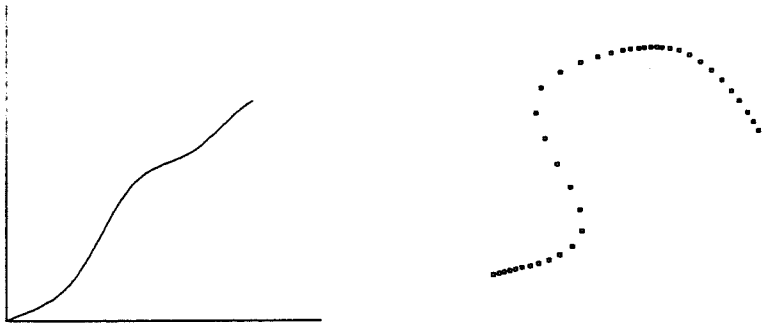


FIG. 3.5 Paramétrisation utilisant l'abscisse curviligne le long de la trajectoire

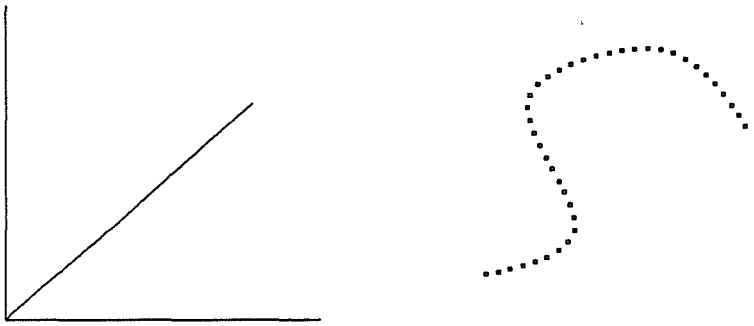


FIG. 3.6 L'abcisse curviligne parcourue est une fonction linéaire du temps

cette recherche est très rapide car effectuée par dichotomie (la propriété de croissance de $f(u)$ est utilisée) — et d'en déduire la valeur de u correspondante. Le calcul de $C(u)$ fournit finalement la valeur du paramètre. Pour résumer, cette valeur du paramètre est obtenue à partir de la formule $P(t) = C(S^{-1}(f(t)))$.

Cette approche nous semble plus naturelle du point de vue de l'utilisateur. Contrairement à la méthode précédente, une courbe $a = f(t)$ linéaire génère la cinétique du mouvement à laquelle on s'attend naturellement : un mouvement où le module des vitesses est constant (figure 3.6).

3.2.3 Spécification de la vitesse en fonction du temps

Cette troisième approche, proposée par Bartels et Hardtke dans [BH89], est basée sur l'utilisation explicite d'une courbe exprimant la vitesse en fonction du temps. Ils présentent cette méthode comme un complément à l'approche exposée au paragraphe 3.2.1. La modification de cette fonction additionnelle influence directement la vitesse, c'est à dire l'échantillonnage, le long de la trajectoire.

La trajectoire spatiale est définie par une courbe $s = s(u)$. La reparamétrisation est effectuée à l'aide de la fonction $u = \Phi(t)$ qui exprime l'évolution de la vitesse au cours du mouvement. Les maxima locaux de Φ correspondent aux pointes de vitesse (les échantillons sur la trajectoire sont les plus espacés) alors que les minima correspondent aux vitesses les plus faibles (les échantillons sur la trajectoire sont les plus proches).

Pour expliquer la méthode, commençons par détailler la solution permettant d'assurer un déplacement à vitesse constante. Cette vitesse est notée K. Effectuer un mouvement à vitesse constante requiert une variation constante de l'abscisse curviligne le long de la trajectoire. En dimension deux, la dérivée de l'abscisse curviligne en fonction de u est donnée par :

$$\frac{ds}{du}\sqrt{\left(\frac{dx}{ds}\right)^2 + \left(\frac{dy}{ds}\right)^2}$$

Afin d'assurer une vitesse constante, cette variation d'abscisse doit être constante, d'où

la condition :

$$\frac{ds}{du} = \frac{K}{\sqrt{\left(\frac{dx}{ds}\right)^2 + \left(\frac{dy}{ds}\right)^2}}$$

En généralisant au cas d'une vitesse $\Phi(u)$ variable, il faut maintenant que s vérifie l'équation différentielle :

$$\frac{ds}{du} = \frac{K\Phi(u)}{\sqrt{\left(\frac{dx}{ds}\right)^2 + \left(\frac{dy}{ds}\right)^2}}$$

Cette équation doit alors être intégrée en prenant comme valeur initiale $s(t_0)$. La constante K est libre et doit être ajustée de manière à ce que l'on retrouve $s(t_n)$ à la fin du processus d'intégration. La méthode d'intégration est du type Runge-Kutta, basée sur une décomposition d'un intervalle (ou un pas d'intégration) en quatre sous-intervalles, ce qui assure une bonne stabilité pour un coût de calcul raisonnable.

La partie la plus coûteuse de la méthode est la détermination de la constante K. K est d'abord encadrée par deux valeurs K_{min} et K_{max} telles que l'intégration utilisant K_{min} fournisse une valeur finale $s(u_n) < s_n$ et celle utilisant K_{max} fournisse une valeur $s(u_n) > s_n$. L'intervalle $[K_{min}, K_{max}]$ est ensuite progressivement réduit jusqu'à ce que la largeur de celui-ci soit inférieure à une valeur prédéfinie suffisamment petite.

Chaque étape de raffinement requiert une intégration complète. Cette méthode est donc relativement coûteuse (selon les auteurs, plusieurs secondes pour une reparamétrisation) et semble donc peut indiquée pour l'interaction. Par contre, elle semble plus intuitive que les deux précédentes puisque l'animateur a la possibilité de manipuler directement la vitesse.

3.2.4 Composition des courbes de paramétrisation

Notons enfin l'approche intéressante exposée dans [SD91b] qui est une extension de la paramétrisation de Steketee et Badler du paragraphe 3.2.1. En fait, plus géréralement, elle peut-être appliquée à chacune des trois paramétrisations présentées ci-dessus.

Pour simplifier, la courbe spatiale est donnée sous la forme $P = C(u)$ et la reparamétrisation sous la forme d'une fonction $u = f(t)$. Le mouvement résultant est obtenu par $P = C(f(t))$.

Le principe de cette approche, totalement différente des précédentes, est de permettre la composition des courbes de reparamétrisation, notées f_i. Le mouvement final est donc dépendant d'un ensemble de n fonctions f_i :

$$P = C(f_1(f_2 \ldots f_{n-1}(f_n(t)) \ldots))$$

Pendant la mise au point de l'animation, l'animateur peut modifier n'importe laquelle des fonctions de paramétrisation, chacune d'elle assurant un effet spécifique. Les auteurs présentent quelques exemples de fonctions particulières :

- Un changement d'échelle qui modifie la durée du mouvement. Ici, la paramétrisation

est définie par une fonction linéaire dont la pente permet soit de comprimer, soit d'étaler la durée du mouvement.

- La création de cycles en utilisant une fonction de paramétrisation en dents de scie.
- Création de décalage temporel
- Etc...

La décomposition de la paramétrisation en une suite de fonctions permet à l'animateur de mieux contrôler la cinétique du mouvement. Les effets induits par chaque courbe f_i sont clairement définis.

Pour assurer cette modularité, l'évaluation de la position sur la trajectoire à un instant précis nécessite l'évaluation en chaîne des fonctions f_i. Aussi, quand l'animation est au point, la composition des fonctions est effectivement évaluée sous la forme d'une seule fonction de paramétrisation.

3.3 Une reparamétrisation des positions basée sur la dynamique

Nous venons de le voir, de nombreuses solutions permettent d'effectuer la reparamétrisation de trajectoires. On peut en dégager quelques points communs. D'abord, elles permettent d'ajuster finement un mouvement; cependant l'habileté de l'animateur doit être mise à contribution. Suivant la forme de la fonction de paramétrisation (en particulier, quand elle utilise l'abscisse curviligne), il est relativement aisé de réaliser un mouvement à vitesse constante. Cependant, la correspondance entre la paramétrisation et la courbe spatiale n'est pas évidente pour l'animateur. Spécifier précisément un freinage sur un morceau de l'animation caractérisé par une forte courbure nécessite de nombreux essais puisqu'il faut arriver à déterminer l'endroit de la courbe de paramétrisation correspondant au virage sur la courbe paramétrée.

L'un des avantages de ces techniques est qu'elles permettent d'utiliser la même paramétrisation pour l'ensemble des courbes définissant le mouvement d'un objet (courbes de position, d'orientation, de couleur, etc...). L'inconvénient est contenu lui aussi dans cette facilité, c'est à dire que toutes entités susceptibles d'être animées sont soumises à la même paramétrisation, ce qui n'est pas toujours souhaitable, mais peut être résolu en définissant une fonction par paramètre.

Par exemple, un solide peut, au même moment, suivre une trajectoire de position très courbée requérant une vitesse réduite et une trajectoire linéaire en orientation (au sens des quaternions) impliquant une vitesse angulaire constante. La solution est évidemment d'associer une courbe de reparamétrisation à chacun des paramètres, ce qui nécessite la définition d'un nombre encore accru de courbes. D'autre part, si l'on se représente bien la notion d'abscisse curviligne dans $\mathcal{R}^3$, l'utilisateur peut-il vraiment maîtriser l'abscisse curviligne de la trajectoire interpolant les orientations (qui plus est, définie dans notre cas

dans un espace de dimension quatre). C'est l'une des raisons justifiant la solution présentée dans les sections à venir (à ce jour, la reparamétrisation n'a été implantée que pour les positions. Le cas des orientations est néanmoins traité).

En plus, toute modification de la fonction de reparamétrisation engendre des perturbations sur le reste du mouvement. Le contrôle n'est donc pas local. Par exemple, dans le cas où la paramétrisation est fonction de la vitesse, si l'utilisateur augmente fortement la vitesse à un endroit de la courbe, toutes les autres vitesses vont être diminuées en conséquence, et ce, de manière à éviter que le mouvement ne dépasse la position finale de la trajectoire.

3.3.1 Principe de la méthode

Notre objectif est de générer des mouvements naturels le long d'une trajectoire définie au préalable. Nous voulons que la méthode soit indépendante du type de courbe utilisé lors de la création de la trajectoire spatiale et ceci afin d'éviter les artefacts cinétiques engendrés par les tangentes. Elle doit aussi demander le minimum d'intervention de la part de l'utilisateur. Enfin elle doit générer un mouvement aussi réaliste que possible.

On veut créer un premier mouvement qui soit aussi réaliste que possible (en assurant par exemple un freinage dans les zones courbes de la trajectoire). Un mouvement à vitesse constante ne le permet pas. Pour répondre à ces conditions, nous avons choisi de minimiser l'ensemble des forces impliquées dans la génération du mouvement. Ce principe est suggéré par le fait que les mouvements réalistes tendent à minimiser l'énergie requise pour le réaliser [WK88].

Selon la loi de Newton $\sum f = ma$, cela revient donc en fait à minimiser l'intégrale des accélérations au cours du mouvement[2]. Nous ne nous préoccupons pas de savoir quelle est l'origine de ces forces. Qu'elles soient dues à la gravité ou à une intervention extérieure, peu importe; ce qui nous préoccupe est qu'elles soient aussi minimes que possible.

La courbe initiale peut-être échantillonnée par un ensemble de points P_i. P_i désigne maintenant la position sur la trajectoire à l'instant t_i. De façon équivalente, l'échantillonnage est défini par la donnée du vecteur à n dimensions $t = (t_1, t_2, \ldots, t_n)$. t_i représente explicitement le temps et varie entre t_1 et t_n, les instants initiaux et finaux de l'animation. Les t_i sont espacés d'une valeur constante, typiquement 1/24 seconde. Pour calculer $P_i = P(t_i)$ sur une courbe constituée de morceaux de courbe d'Hermite, il suffit de déterminer le morceau de courbe concerné et, à partir de t_i, la valeur du paramètre u sur ce tronçon et enfin la valeur $P(u)$ sur ce dernier. Notre méthode consiste à ajuster progressivement les t_i, c'est à dire les positions échantillonnées sur la trajectoire.

Puisque les paramètres à ajuster sont des instants discrets, nous allons utiliser une technique d'optimisation. Celle-ci va nous permettre de minimiser progressivement la valeur du critère que nous nous sommes fixé, à savoir la somme des forces intervenant au cours du mouvement. Plus précisément, nous cherchons à minimiser l'amplitude des forces.

[2]La masse n'est pas prise en compte lors du processus d'optimisation, les objets gardant une masse constante au cours du mouvement

Nous allons donc minimiser la somme du carré des accélérations discrétisées. Prendre les carrés des accélérations permet d'assurer que le critère est toujours positif et donc qu'il admet un minimum. Le critère à minimiser prend donc la forme suivante :

$$C = \sum_{1 \leq i \leq n} a_i^2$$

a_i désigne l'accélération à l'instant t_i. Classiquement, l'évaluation de l'accélération est réalisée grâce à la méthode des différences finies. L'intervalle de temps entre deux images est constant et égal à h. D'abord, la vitesse v_i en t_i, est donnée par :

$$v_i = \frac{P_{i+1} - P_i}{h}$$

d'où l'accélération

$$\begin{aligned} a_i &= \frac{v_i - v_{i-1}}{h} \\ &= \frac{P_{i-1} - 2P_i + P_{i+1}}{h^2} \end{aligned} \tag{3.1}$$

En utilisant l'expression (3.1), nous avons :

$$\begin{aligned} h^4 a_i^2 &= (P_{i+1} - 2P_i + P_{i-1})^2 \\ &= 4P_i^2 + P_{i+1}^2 + P_{i-1}^2 + 2P_{i+1}P_{i-1} - 4P_i(P_{i+1} + P_{i-1}) \end{aligned} \tag{3.2}$$

Le problème revient à trouver un vecteur t à n dimensions tel que $C(t)$ soit minimal (voir annexe C). Nous utilisons un processus itératif consistant à améliorer progressivement le vecteur t initial. A une étape donnée, t est transformé en un vecteur $t^* = t + p$ qui doit vérifier $C(t^*) \leq C(t)$. De manière à améliorer la vitesse de convergence, nous utilisons un développement en série de Taylor d'ordre deux du critère C :

$$C(t+p) = C(t) + g^T(t)p + \frac{1}{2}p^T H(t)p$$

g représente le gradient de C. C'est un vecteur colonne dont chaque élément contient la dérivée de C par rapport au paramètre correspondant :

$$g_i = \frac{\partial C}{\partial t_i}$$

H est le hessien de C, une matrice carrée dont chaque ligne contient les dérivées du gradient par rapport à chaque paramètre :

$$H_{i,j} = \frac{\partial^2 C}{\partial t_i \partial t_j}$$

On montre (voir annexe C) que de manière à assurer la décroissance du critère, il faut annuler la dérivée par rapport à p du critère. Cette dérivée est ici égale à $g(t) + H(t)p$. Il nous faut donc résoudre le système linéaire suivant :

$$g(t) + H(t)p = 0 \tag{3.3}$$

L'algorithme simplifié permettant d'atteindre la solution peut se résumer ainsi :

```
{   estimer les valeurs initiales de t
    faire {
        évaluer H(t) et g(t)
        résoudre le système linéaire H(t)p = −g(t)
        t = t + p
    }jusqu'à ce que le minimum soit atteint
}
```

L'estimation du vecteur initial se fait simplement en prenant un échantillonnage régulier le long de la courbe : $t_i = \frac{t_n - t_1}{n} i$.

Il reste maintenant à fournir de plus amples détails sur le calcul effectif du gradient et de la matrice hessienne de C. Nous présentons aussi un algorithme permettant, compte-tenu des particularités de la matrice hessienne, d'effectuer la résolution du système linéaire en temps proportionnel à la taille du problème.

3.3.2 Calcul des dérivées premières et secondes

Afin d'éclaircir les calculs ultérieurs, commençons par isoler de C les termes où intervient le paramètre t_i, que nous noterons C_i. Au vu de l'équation (3.2), seules les expressions a_i^2, a_{i-1}^2, a_{i+1}^2 peuvent contenir des termes t_i :

$$\begin{aligned} a_{i-1}^2 &= 4P_{i-1}^2 + P_i^2 + P_{i-2}^2 + 2P_iP_{i-2} - 4P_{i-1}(P_i + P_{i-2}) \\ a_i^2 &= 4P_i^2 + P_{i+1}^2 + P_{i-1}^2 + 2P_{i+1}P_{i-1} - 4P_i(P_{i+1} + P_{i-1}) \\ a_{i+1}^2 &= 4P_{i+1}^2 + P_{i+2}^2 + P_i^2 + 2P_{i+2}P_i - 4P_{i+1}(P_{i+2} + P_i) \end{aligned}$$

d'où

$$C(t_i) = C_i = 6P_i^2 - 8P_i(P_{i-1} + P_{i+1}) + 2P_i(P_{i-2} + P_{i+2})$$

La dérivée première de C par rapport à t_i, en traitant toutes les autres variables comme des constantes est :

$$g_i = \frac{\partial C_i}{\partial t_i} = 12P_iP_i' - 8P_i'(P_{i-1} + P_{i+1}) + 2P_i'(P_{i-2} + P_{i+2})$$

La matrice hessienne contenant les dérivées secondes de C est obtenue en dérivant g_i par rapport à une variable t_j :

$$\frac{\partial^2 C_i}{\partial t_i \partial t_j} = \begin{cases} 2P_i'P_{i-2}' & j = i-2 \\ -8P_i'P_{i-1}' & j = i-1 \\ 12(P_i'^2 + P_iP_i'') - 8P_i''(P_{i-1} + P_{i+1}) + 2P_i''(P_{i-2} + P_{i+2}) & j = i \\ -8P_i'P_{i+1}' & j = i+1 \\ 2P_i'P_{i+2}' & j = i+2 \\ 0 & \text{sinon} \end{cases} \tag{3.4}$$

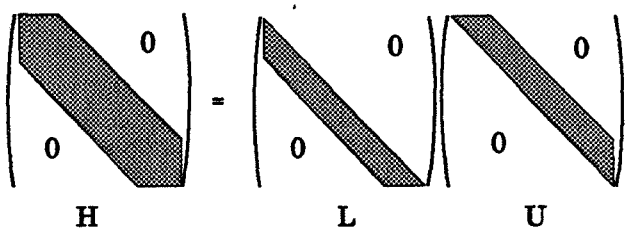


FIG. 3.7 Décomposition de la matrice hessienne

Les dérivées P' et P'' sont directement déduites de l'expression initiale de l'interpolation d'Hermite (équation 1.17). Il suffit de substituer au vecteur contenant les puissances successives du paramètre, les dérivées premières et secondes de ce vecteur, à savoir $\begin{pmatrix} 3u^2 & 2u & 1 & 0 \end{pmatrix}$ dans le premier cas et $\begin{pmatrix} 6u & 2 & 0 & 0 \end{pmatrix}$ dans le second.

On constate facilement que la matrice hessienne est une matrice penta-diagonale. L'occupation en place mémoire peut, de ce fait, être substantiellement réduite, ce qui est particulièrement important quand le nombre de variables est élevé. Seuls les éléments non nuls de cette matrice ont un intérêt. Ils sont stockés dans un tableau à n lignes et 5 colonnes, chacune des lignes contenant les éléments significatifs de la diagonale.

Nous allons aussi tirer parti de cette structure particulière lors de la résolution du système linéaire $Hp = -g$ qui va maintenant être détaillée.

3.4 Résolution du système linéaire

L'algorithme que nous utilisons est basé sur une décomposition LU de la matrice H. Plus précisément, le système $Hp = -g$ est mis sous la forme $LUp = -g$, où L est une matrice triangulaire inférieure et U une matrice triangulaire supérieure. H étant penta-diagonale, les matrices triangulaires ne contiennent chacune que trois diagonales significatives; tous les autres éléments sont nuls (figure 3.7). Ce dernier système est résolu en deux étapes. Pour commencer, on résout – par substitution avant – le système $Ly = -g$, ensuite le système $Up = y$ par substitution arrière ce qui nous donne la solution recherchée.

Cette méthode très générale, dite méthode de Crout, a été adaptée de manière à tirer parti de la structure de H. Nous nous sommes inspiré d'un algorithme basé sur ce principe [PFTV92] et spécifique au cas des matrices tridiagonales.

La décomposition LU et la résolution du système triangulaire inférieur sont effectuées en une seule étape prenant un temps proportionnel à la taille de la matrice. Le stockage de L n'est donc pas nécessaire. Par contre, la matrice supérieure U doit être stockée temporairement dans un tableau de taille $n \times 3$ afin d'être utilisée pendant l'étape de subsitution arrière, qui elle aussi s'effectue en temps proportionnel à n.

Notons que grâce aux valeurs spéciales que prennent les éléments de la matrice hessienne, il n'est pas utile d'effectuer de pivotage. Une condition suffisante pour éviter le pivotage est que la matrice H soit à diagonale fortement dominante. Dans l'application

qui nous intéresse, nous n'avons pu démontrer cette propriété dans le cas général. En pratique, cela n'a jamais posé de problèmes. C'est un phénomène qui se produit fréquemment pour des matrices issues de problèmes mécaniques ou physiques.

3.5 Reparamétrisation des orientations

Dans cette partie, nous allons adapter la méthode précédente à la reparamétrisation des orientations interpolées (à ce jour, cette partie n'a pas été implantée). Comme cela a été souligné plus haut, il est difficile d'appréhender aisément la notion d'abscisse curviligne le long d'une trajectoire interpolant des orientations, donc de définir et contrôler une courbe de reparamétrisation dans le cas des orientations. L'intérêt d'une méthode automatique nous en semble renforcé.

3.5.1 Principe

La méthode que nous nous proposons de développer est identique dans le principe à la précédente. Cette fois, nous utilisons la loi de Newton appliquée aux orientations :

$$T = I\dot{\omega}$$

où

- T représente la somme des couples appliqués au solide. Encore une fois, on ne se préoccupe pas de leur origine.
- I est la matrice d'inertie du solide. Elle doit être définie dans le repère local du solide. Malheureusement, celui-ci ne se situe pas forcément au centre de masse de l'objet. I n'est donc pas forcément diagonale, ce qui compliquera les calculs ultérieurs [3].
- $\dot{\omega}$ représente l'accélération angulaire du solide.

[3] En pratique, nous n'utilisons pas la matrice d'inertie exacte de l'objet considéré. Celle-ci est approximée par la matrice d'inertie de la boîte englobante de l'objet. En prenant le centre de la boîte englobante – de coordonnées $G = \begin{pmatrix} x_G & y_G & z_G \end{pmatrix}$ dans le repère local de l'objet – comme centre de masse et si a, b et c désignent la longueur de chacun des trois côtés de la boîte englobante le long des trois axes, la matrice d'inertie est donnée par :

$$I = \frac{1}{12}m \begin{pmatrix} b^2+c^2 & 0 & 0 \\ 0 & a^2+c^2 & 0 \\ 0 & 0 & a^2+b^2 \end{pmatrix} + m \begin{pmatrix} y_G^2+z_G^2 & -x_G y_G & -x_G z_G \\ -x_G y_G & x_G^2+z_G^2 & -y_G z_G \\ -x_G z_G & -y_G z_G & x_G^2+y_G^2 \end{pmatrix}$$

Le calcul exact de cette matrice est cependant possible, même pour des objets modélisés par arbre de construction. Il suffit de décomposer l'objet en blocs élémentaires et d'appliquer la formule discrétisée du calcul général d'une matrice d'inertie. Cela peut en particulier être réalisé à l'aide d'un algorithme de tracé de rayons [Roe93]; simplement, les informations de rendu sont remplacées par des informations sur la masse des petits blocs (chacun d'eux représentant l'intersection entre un rayon et l'objet)

Suivant le même principe que précédemment, on s'attache maintenant à minimiser l'amplitude des couples impliqués dans la génération du mouvement. Pour cela, on définit le critère représentant la somme discrétisée des carrés des couples intervenant à chaque intervalle de temps :

$$C = \sum_{1 \leq i \leq n} (I\dot{\omega}_i)^2$$

$\dot{\omega}_i$ est l'accélération angulaire à l'instant t_i. A nouveau, la méthode des différences finies — plus l'emploi des logarithmes de quaternions — nous permet d'évaluer cette quantité. D'abord, la vitesse angulaire peut être représentée par l'angle θ_i parcouru entre l'orientation q_i et l'orientation q_{i+1} autour d'un axe unitaire v_i en un temps h, c'est à dire par :

$$\omega_i = \frac{\theta_i v_i}{h}$$

Or, $\frac{\theta_i v_i}{2}$ est le logarithme du quaternion qui transforme q_i en q_{i+1}. Nous avons donc la relation suivante :

$$\omega_i = \frac{\log q_{i+1} - \log q_i}{2h}$$

L'accélération angulaire à l'instant t_i est alors simplement donnée par :

$$\dot{\omega}_i = \frac{\omega_i - \omega_{i-1}}{h} = \frac{\log q_{i+1} - 2\log q_i + \log q_{i-1}}{2h^2}$$

Le gradient de C ainsi que sa matrice hessienne sont calculés de manière analogue au paragraphe 3.3.2. Les logarithmes des quaternions interpolés ($\log q(t_i)$) sont substitués aux positions interpolées $p(t_i)$. Simplement, les expressions sont plus compliquées.

Le fait important est que la matrice hessienne demeure penta-diagonale, ce qui nous assure de conserver de bonnes performances pendant l'exécution de l'algorithme.

3.5.2 Calcul des dérivées premières et secondes

Le critère à minimiser peut aussi s'écrire

$$C = \sum_{i=1}^{n} \omega_i^T I^T I \omega_i$$

Remarquons tout de suite que le produit $I^T I$ peut être évalué une fois pour toute. Le résultat est conservé dans une matrice, symétrique par construction, notée I^2 dont les éléments vérifient :

$$I^2_{i,j} = \sum_{k=1}^{3} I_{k,i} I_{k,j}$$

D'autre part, nous allons être amené à traiter les composantes des vecteurs $\log \omega_i$ séparément. La notation $\omega_{i,j}$ désigne le $j^{\text{ème}} (1 \leq j \leq 3)$ élément du vecteur ω_i. De même $\log q_{i,j}$ désigne le $j^{\text{ème}}$ élément de $\log q_i$.

Afin de déterminer l'expression des dérivées du critère à minimiser, commençons par détailler le terme $\omega_i^T I^2 \omega_i$ de C. Nous le noterons C_i.

$$\begin{aligned} C_i &= I_{1,1}^2\omega_{i,1}^2 + I_{2,2}^2\omega_{i,2}^2 + I_{3,3}^2\omega_{i,3}^2 \\ &\quad +2I_{1,2}^2\omega_{i,1}\omega_{i,2} + 2I_{1,3}^2\omega_{i,1}\omega_{i,3} + 2I_{2,3}^2\omega_{i,2}\omega_{i,3} \end{aligned}$$

En éliminant tous les termes où n'intervient pas t_i (ou un produit avec t_i), on a :

$$\begin{aligned} C_i &= I_{1,1}^2\left(4\log q_{i,1}^2 - 4\log q_{i,1}\left(\log q_{i+1,1} + \log q_{i-1,1}\right)\right) \\ &\quad +I_{2,2}^2\left(4\log q_{i,2}^2 - 4\log q_{i,2}\left(\log q_{i+1,2} + \log q_{i-1,2}\right)\right) \\ &\quad +I_{3,3}^2\left(4\log q_{i,3}^2 - 4\log q_{i,3}\left(\log q_{i+1,3} + \log q_{i-1,3}\right)\right) \\ &\quad +2I_{1,2}^2(-2\log q_{i,1}\left(\log q_{i+1,2} + \log q_{i-1,2}\right) - 2\log q_{i,2}\left(\log q_{i+1,1} + \log q_{i-1,1}\right) \\ &\qquad +4\log q_{i,1}q_{i,2}) \\ &\quad +2I_{1,3}^2(-2\log q_{i,1}\left(\log q_{i+1,3} + \log q_{i-1,3}\right) - 2\log q_{i,3}\left(\log q_{i+1,1} + \log q_{i-1,1}\right) \\ &\qquad +4\log q_{i,1}q_{i,3}) \\ &\quad +2I_{2,3}^2(-2\log q_{i,2}\left(\log q_{i+1,3} + \log q_{i-1,3}\right) - 2\log q_{i,3}\left(\log q_{i+1,2} + \log q_{i-1,2}\right) \\ &\qquad +4\log q_{i,2}q_{i,3}) \end{aligned}$$

Ce qui donne, en notant $A_k = 4\log q_{i,k} - 4\log q_{i+1,k} - 4\log q_{i-1,k}$:

$$\begin{aligned} C_i &= \log q_{i,1}\left(I_{1,1}^2A_1 + I_{1,2}^2A_2 + I_{1,3}^2A_3\right) \\ &\quad +\log q_{i,2}\left(I_{1,2}^2A_1 + I_{2,2}^2A_2 + I_{2,3}^2A_3\right) \\ &\quad +\log q_{i,3}\left(I_{1,3}^2A_1 + I_{2,3}^2A_2 + I_{3,3}^2A_3\right) \end{aligned}$$

et, de manière plus condensée :

$$C_i = \sum_{j=1}^{3}\log q_{i,j}\sum_{k=1}^{3}I_{j,k}^2A_k \tag{3.5}$$

D'autre part, les membres $\omega_{i+1}^T I^2\omega_{i+1}$ et $\omega_{i-1}^T I^2\omega_{i-1}$ de la somme C contiennent eux-aussi des termes dépendants de t_i. En appliquant une méthode de calcul similaire à ce qui précède et en ne conservant toujours que les éléments dépendants de t_i, on obtient :

$$C_{i-1} = \sum_{j=1}^{3}\log q_{i,j}\sum_{k=1}^{3}I_{j,k}^2B_k \tag{3.6}$$

$$C_{i+1} = \sum_{j=1}^{3}\log q_{i,j}\sum_{k=1}^{3}I_{j,k}^2C_k \tag{3.7}$$

avec

$$\begin{aligned} B_k &= \log q_{i,k} - 4\log q_{i-1,k} + 2\log q_{i-2,k} \\ C_k &= \log q_{i,k} - 4\log q_{i+1,k} + 2\log q_{i+2,k} \end{aligned}$$

En additionnant les expressions (3.6), (3.5) et (3.7), les termes de C où apparaissent des termes en t_i s'écrivent :

$$C_i = \sum_{j=1}^{3} \log q_{i,j} \sum_{k=1}^{3} I_{j,k}^2 D_k$$

avec

$$D_k = 2\log q_{i-2,k} - 8\log q_{i-1,k} + 6\log q_{i,k} - 8\log q_{i+1,k} + 2\log q_{i+2,k}$$

Evaluons maintenant la dérivée de cette expression par rapport à t_i :

$$\begin{aligned} \frac{\partial C}{\partial t_i} &= \log q'_{i,1} I_{1,1}^2 D_1 + 6\log q_{i,1} I_{1,1}^2 \log q'_{i,1} + \log q'_{i,1} I_{1,2}^2 D_2 \\ &\quad +6\log q_{i,1} I_{1,2}^2 \log q'_{i,2} + \log q'_{i,1} I_{1,3}^2 D_3 + 6\log q_{i,1} I_{1,3}^2 \log q'_{i,3} \\ &\quad + \log q'_{i,2} I_{1,2}^2 D_1 + 6\log q_{i,2} I_{1,2}^2 \log q'_{i,2} + \log q'_{i,2} I_{2,2}^2 D_2 \\ &\quad +6\log q_{i,2} I_{2,2}^2 \log q'_{i,2} + \log q'_{i,2} I_{2,3}^2 D_3 + 6\log q_{i,2} I_{2,3}^2 \log q'_{i,3} \\ &\quad + \log q'_{i,3} I_{1,3}^2 D_1 + 6\log q_{i,3} I_{1,3}^2 \log q'_{i,3} + \log q'_{i,3} I_{2,3}^2 D_2 \\ &\quad +6\log q_{i,3} I_{2,3}^2 \log q'_{i,3} + \log q'_{i,3} I_{3,3}^2 D_3 + 6\log q_{i,3} I_{3,3}^2 \log q'_{i,3} \\ &= \sum_{j=1}^{3} \log q'_{i,j} \sum_{k=1}^{3} I_{j,k}^2 \left(D_k + \log q_{i,k}\right) \end{aligned}$$

Finalement,

$$\frac{\partial C}{\partial t_i} = \sum_{j=1}^{3} \log q'_{i,j} \sum_{k=1}^{3} I_{j,k}^2 E_k$$

avec

$$E_k = 2\log q_{i-2,k} - 8\log q_{i-1,k} + 12\log q_{i,k} - 8\log q_{i+1,k} + 2\log q_{i+2,k}$$

De même que pour les positions, les dérivées premières et secondes, $\log q'_i$ et $\log q''_i$, sont obtenues en remplaçant dans l'équation (1.18) le vecteur contenant les puissances du paramètre par les dérivées premières et secondes de ce vecteur.

L'expression du gradient contient des termes dépendants, outre du paramètre t_i, des paramètres t_{i-2}, t_{i-1}, t_{i+1} et t_{i+2}. Le hessien est donc à nouveau une matrice pentadiagonale. Sans détailler les calculs, simples mais lourds, cette matrice est donnée par

$$\frac{\partial^2 C_i}{\partial t_i \partial t_j} = \begin{cases} 2\sum_{l=1}^{3} \log'_{i,l} \sum_{k=1}^{3} \log'_{i-2,k} I_{l,k}^2 & j = i-2 \\ -8\sum_{l=1}^{3} \log'_{i,l} \sum_{k=1}^{3} \log'_{i-1,k} I_{l,k}^2 & j = i-1 \\ \sum_{l=1}^{3} \left(\log q''_{i,l} \sum_{k=1}^{3} I_{l,k}^2 E_k + 12\log q'_{i,l} \sum_{k=1}^{3} I_{l,k}^2 \log'_{i,k}\right) & i = j \\ -8\sum_{l=1}^{3} \log'_{i,l} \sum_{k=1}^{3} \log'_{i+1,k} I_{l,k}^2 & j = i+1 \\ 2\sum_{l=1}^{3} \log'_{i,l} \sum_{k=1}^{3} \log'_{i+2,k} I_{l,k}^2 & j = i+2 \\ 0 & \text{sinon} \end{cases} \tag{3.8}$$

3.6 Contrôle de la cinétique du mouvement

L'algorithme de minimisation fournit automatiquement un mouvement satisfaisant. Il est malgré tout important de laisser à l'utilisateur la possibilité d'intervenir sur le résultat. Deux techniques de contrôle sont possibles.

La première solution revient à transformer le résultat sous la forme d'une courbe de reparamétrisation telle que celle qui a été décrite en introduction (paragraphes 3.2.1, 3.2.2 et 3.2.3). Il suffit alors d'utiliser des techniques de paramétrisation plus classiques afin d'ajuster finement la cinétique du mouvement. La transformation du vecteur t minimisant le critère en courbe de paramétrisation est facile et, suivant le type de courbe, s'obtient de la façon suivante :

- Courbe $u = f(t)$ (paragraphe 3.2.1). La courbe f est la fonction discrète, éventuellement lissée par une spline d'interpolation, qui aux valeurs t_i initiales (équitablement réparties) associe les valeurs de t_i, résultats de l'algorithme de minimisation.
- Courbe $a = f(t)$ donnant l'abscisse curviligne en fonction du temps (§ 3.2.2). A chaque valeur t_i, on associe la valeur de la distance entre $P(t_i)$ et $P(t_{i-1})$:

  $$f(t_i) = \text{distance}(P_i, P_{i+1}) + f(t_{i-1})$$

  Là aussi, f est forcément discrète mais peut être lissée.
- Courbe donnant la vitesse en chaque instant (§ 3.2.3). La méthode des différences finies est ici utilisée afin d'approximer la valeur de la vitesse à chaque instant t_i. A nouveau, la courbe peut-être lissée.

Dans ces différentes applications, notre paramétrisation a pour intérêt de trouver une première approximation du mouvement. La qualité du mouvement généré permet à l'utilisateur de ne se consacrer qu'à de petits réglages qui donneront la touche finale à l'animation.

La deuxième possibilité de contrôle consiste à influer directement sur le comportement de l'algorithme. Ceci se fait par la spécification de contraintes additionnelles (en plus des contraintes maintenant le mouvement sur la trajectoire initiale). Deux types de contraintes sont possibles :

- Contraintes de vitesses : celles-ci ne peuvent pour l'instant être fournies qu'aux extrémités de la trajectoire.
- Contraintes de positions : il est possible d'imposer à l'objet à animer de passer par certains points de la trajectoire en des instants déterminés. La méthode consiste à décomposer le mouvement initial en plusieurs mouvements dont la concaténation représente le mouvement global. Par exemple, si on impose le passage par le point P_c à l'instant t_c, l'animation de t_1 à t_n est décomposée en deux : un mouvement de t_1 à t_c entre $P(t_1)$ et P_c et un mouvement de t_c à t_n entre P_c et $P(t_n)$.

Une extension intéressante, permettant de généraliser ces deux types de contrainte, serait d'utiliser un algorithme d'optimisation sous contrainte.

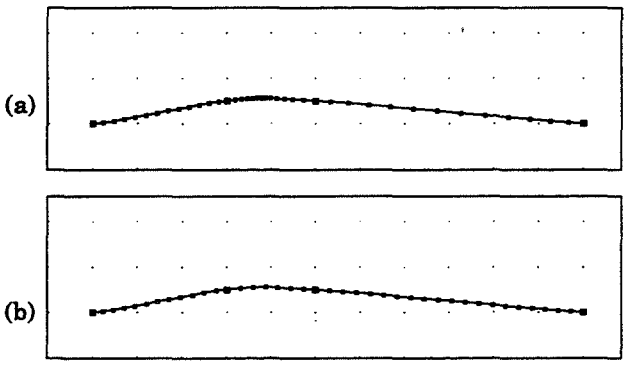


FIG. 3.8 Répartition des échantillons

3.7 Exemples

La paramétrisation de deux trajectoires différentes est exposée. La première (figure 3.8) est quasiment plate mais les points de contrôle sont irrégulièrement espacés. Sur la partie (a), les points sont équitablement répartis entre chaque morceau de spline; l'écartement entre les échantillons varie avec celui des points de contrôle. Sur la partie (b), la reparamétrisation a permis de répartir régulièrement les échantillons.

La deuxième trajectoire est plus complexe (figure 3.9). La partie (a) montre le mouvement initial. Après la reparamétrisation (b), les échantillons se répartissent le long de la trajectoire. On peut constater l'effet de freinage dans la partie la plus courbe et d'accélération dans la portion presque linéaire du parcours. Sur la figure de droite, la vitesse initiale (en bas à gauche) a été augmentée.

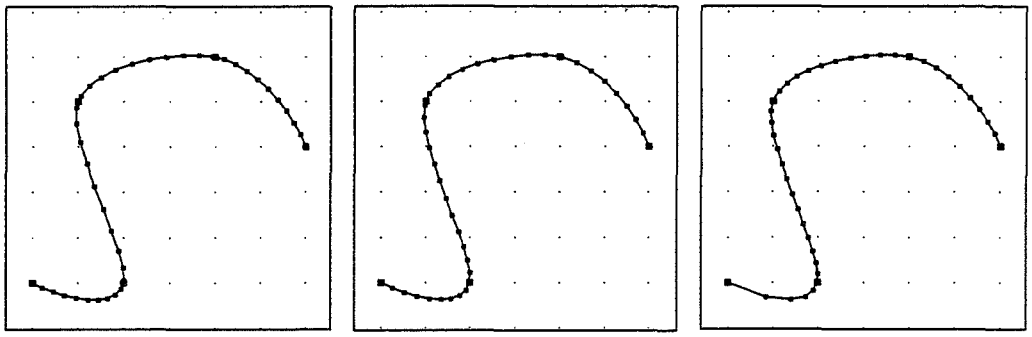
Toutes les courbes sont dotées du même nombre d'échantillons. Pour que l'objet aille globalement plus vite (qu'il ait une vitesse moyenne plus élevée), il faut utiliser plus d'échantillons.

3.8 Conclusion

Nous venons de présenter une méthode de paramétrisation dont l'une des particularités est qu'elle requiert peu de travail de la part de l'utilisateur. Grâce à l'utilisation de quelques lois élémentaires de la mécanique, nous avons obtenu des résultats intéressants. On peut noter une certaine analogie avec l'approche présentée dans [PF88] basée sur la notion de champs générateurs du mouvement.

Ces méthodes permettent de résoudre les problèmes que posent généralement les splines cubiques concernant la cinétique du mouvement, et en particulier la sensibilité du mouvement aux distances entre les points de contrôle.

D'autre part, comme avec les splines de degré plus élevé, le mouvement se ralentit dans les courbes et s'accélère à l'entrée des portions rectilignes. En fait, nous minimisons des accélérations alors que les splines minimisent la courbure; ce qui, somme toute, n'est pas



(a) (b) (c)

FIG. 3.9 Influence de la courbure (b) et de la vitesse initiale (c)

très différent. Cependant, contrairement aux splines, l'allure de la trajectoire et la cinétique du mouvement le long de celle-ci sont traitées de manière bien distincte. Avec les splines habituelles, les tangentes et les points de contrôle sont utilisés comme des contraintes. Dans notre approche, ce sont les modules de vitesse, les points caractéristiques et la trajectoire spatiale qui jouent ce rôle.

La prise en compte de ces contraintes (exceptée la trajectoire spatiale) nécessite la spécification d'une méthodologie afin de les intégrer dans la partie interactive précédemment exposée. Cela n'a pas encore été réalisé; en tous cas, nous ne pouvons pas spécifier les vitesses à l'aide des tangentes, ces dernières étant déjà affectées au contrôle des trajectoires spatiales.

Enfin, il est tout à fait possible d'utiliser notre méthode de reparamétrisation comme un pré-traitement permettant de générer automatiquement des courbes de reparamétrisation acceptables, que l'animateur pourra ensuite éditer à sa guise.

Chapitre 4

Modélisation interactive des cylindres généralisés

4.1 Introduction

La construction d'un cylindre généralisé se fait en déplaçant un élément plan pour produire un élément de dimension trois. Le déplacement, appelé aussi balayage, peut être une rotation, une translation, une courbe, une déformation ou une combinaison des précédentes. De nombreux termes, suivant le type de mouvement, désignent cette classe d'objets dont le plus est général est l'extrusion. D'autres termes tels que cône généralisé, objet de révolution, plus les divers termes anglais s'appliquent à certaines constructions spécifiques. Nous adopterons ici celui de cylindre généralisé pour des raisons détaillées plus loin.

En fait, l'animation d'un objet bi-dimensionnel le long d'une trajectoire crée implicitement une extrusion. Il suffit de relier entre elles les diverses occurrences de l'objet au cours du temps. Il nous a donc semblé intéressant d'appliquer quelques unes des techniques précédentes, à la modélisation de tels objets.

Dans le premier chapitre, nous nous sommes plus particulièrement préoccupés du contrôle des orientations lors d'une animation. Nous étendrons la méthode pour tenir compte des positions interpolées (elles déterminent le balayage en translation) avec une méthodologie identique. Ensuite, après un rappel sur les extrusions, nous traiterons des techniques que nous avons été amenés à développer pour prendre en compte certaines caractéristiques de notre modèle (quaternions, interpolation d'Hermite).

4.2 Prise en compte des positions

4.2.1 Interpolation générale

Dans le premier chapitre, nous avons donné des solutions pour le contrôle interactif des orientations. Les repères locaux représentant les orientations-clés q_i étaient tous situés à l'origine. Les positions de ces repères locaux vont maintenant être mobiles; nous les noterons P_i. Comme pour les splines d'orientation, les positions-clés seront interpolées à l'aide des polynômes d'Hermite. On appelle alors un *paramètre clé* la réunion d'une orientation et d'une position sous la forme du vecteur de dimension six[1] :

$$K_i = \begin{pmatrix} P_i & \log q_i \end{pmatrix}$$

Les paramètres interpolés $K_i(u)$ sur un morceau de spline d'Hermite sont alors calculés comme suit :

$$K(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} M_{spline} \begin{pmatrix} P_i & \log q_i \\ R_i & \log qr_i \\ P_{i+1} & \log q_{i+1} \\ G_{i+1} & \log qg_i \end{pmatrix} \quad 0 \leq u \leq 1$$

Les orientations sont retrouvées en appliquant l'exponentielle aux logarithmes des quaternions. La courbe globale $K(t)$, avec $0 \leq t \leq n$, est composée des n morceaux $K_i(u)$ où i = partie entière(t) et $u = t - i$.

4.2.2 Contrôle commun des orientations et des positions

Pour tenir compte des positions-clés, il suffit de laisser le loisir à l'utilisateur de manipuler les positions de chaque repère local. Comme pour les orientations, la même souplesse est offerte sur le contrôle des positions interpolées grâce à l'utilisation des tangentes à la courbe. Par la suite, celles-ci seront appelées *tangentes planes* par opposition aux tangentes sphériques. Là encore, nous utilisons l'interface du chapitre 2, tant pour assurer le déplacement tridimensionnel des positions-clés que le réglage des tangentes.

Un des grands avantages de notre méthode est que l'utilisateur peut contrôler sur la même fenêtre les deux interpolations. Pour cela, nous utilisons deux courbes. La première interpole classiquement les centres des repères locaux. Par contre, la courbe des orientations dépend de la trajectoire des positions. Elle représente maintenant le chemin suivi par un point de l'objet soumis à la fois aux positions et aux orientations interpolées. A tout moment, l'utilisateur peut sélectionner un autre point de l'objet. Sur la figure (4.1), on distingue les deux courbes munies de leurs tangentes respectives. La courbe de position passe par le centre du **T**, la courbe d'orientation par sa base. La figure (4.2) donne l'animation résultante.

[1] Il est évident que toute autre quantité susceptible d'être animée – tel qu'un vecteur 3D représentant la couleur de l'objet ou un scalaire représentant sa taille – pourrait être ajoutée à K_i

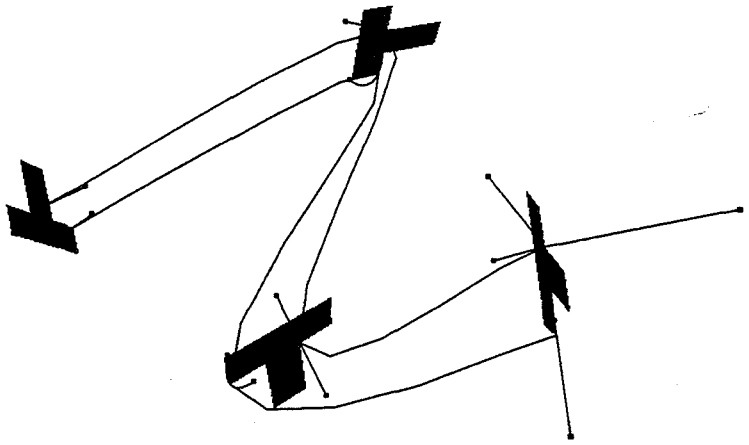


FIG. 4.1 Courbes de position et d'orientation.

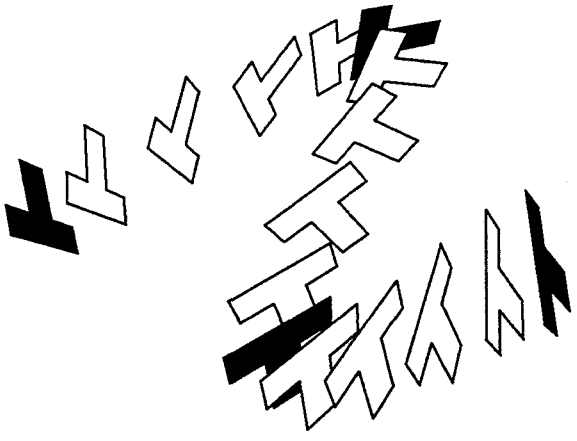


FIG. 4.2 Animation des positions et des orientations.

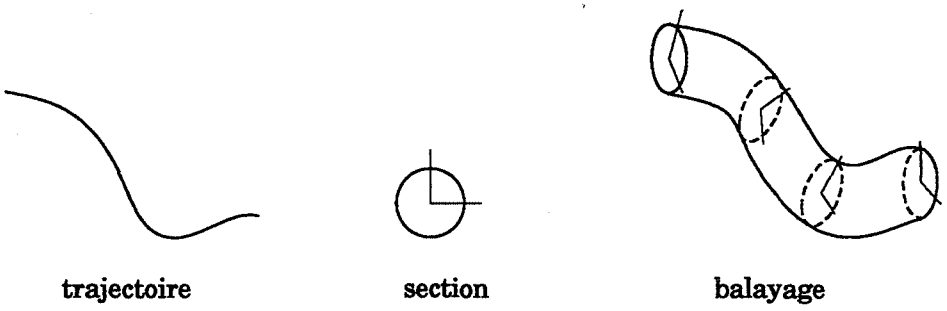


FIG. 4.3 Exemple d'objet obtenu par extrusion.

Remarque

Les trajectoires d'orientation ne se trouvent plus sur une sphère puisqu'elles dépendent maintenant de la position interpolée. Par contre, les tangentes sphériques demeurent des arcs de cercle – et non une composition entre les tangentes sphériques et planes. Cette présentation reflète mieux les effets de rotation engendrés par ces tangentes.

4.3 Les cylindres généralisés

Un cylindre généralisé est défini par la donnée d'une section, d'une trajectoire et d'une courbe d'orientation [RM88, Klo86, CH89, AA92]. Le volume est engendré par le déplacement de la section le long de la trajectoire (figure 4.3). Quand le contour de la section n'est pas fermé, l'objet généré n'est que surfacique. Par ailleurs, la courbe d'orientation permet de spécifier l'évolution d'un angle de rotation au cours du déplacement. Cette rotation est appliquée à la section par rapport à un axe tangent à la trajectoire. Elle permet de créer des effets de torsion sur l'objet. D'autres courbes, telle qu'une fonction exprimant la variation de l'échelle (on parle alors de cône généralisé) ou de la forme de la section le long de la trajectoire, peuvent être définies.

Si la trajectoire est définie par une courbe paramétrée $T(t)$, la section par une autre courbe paramétrée $S(s)$, la surface du cylindre généralisé est définie par l'équation :

$$CG(t,s) = T(t) + S_x(s)t(t) + S_y(s)n(t)$$

où $t(t)$, $n(t)$ ainsi qu'un troisième vecteur $b(u)$ représentent un repère local lié à la trajectoire (voir § 4.5).

D'autre part, l'ensemble des variations en taille, forme, orientation que subit la section lors du déplacement le long de la trajectoire peut être exprimé à l'aide d'une matrice de transformation homogène $R(u)$. Celle-ci permet de déterminer la valeur de la section pour chaque valeur de t :

$$S(t) = R(t)S(s)$$

4.4 Notre approche

Dans les systèmes habituels de modélisation des objets obtenus par extrusion, l'interaction consiste principalement à utiliser un éditeur de courbes pour définir une à une la trajectoire, l'orientation, la section constituant le cylindre généralisé. A partir des idées du paragraphe 4.2.2, nous pouvons envisager le contrôle des différents paramètres au travers d'une seule fenêtre, contenant toutes les informations géométriques du solide. L'intérêt n'est pas seulement d'économiser de la place sur l'écran, mais surtout d'appréhender et de contrôler directement la forme complète du solide.

En fait, nous nous sommes limités au contrôle de la trajectoire et de l'orientation de la section. Il serait néanmoins facile de manipuler d'autres paramètres grâce aux possibilités d'interaction dont nous disposons. Par exemple, pour définir l'échelle de la section en chaque paramètre clé, il suffirait de permettre à l'utilisateur de tirer sur un des points de la section et de le déplacer dans le plan qu'elle occupe. Une même méthode peut servir à modifier la forme de la section, en restant toujours dans la même zone de dialogue.

Contrairement à l'animation, les auteurs dans ce domaine se restreignent à une seule rotation (un seul degré de liberté en rotation autour de l'axe tangent à la trajectoire). Peut être est-ce dû à la complexité de contrôler, voire simplement d'interpoler une trajectoire en orientation. De notre côté, disposant d'une technique performante pour contrôler les trajectoires d'orientations, nous avons décidé de n'imposer aucune limitation sur la courbe d'orientation (nous verrons dans le paragraphe 4.6.2 comment sont résolus les problèmes d'homogénéité résultants).

Pour créer des cylindres généralisés, nous n'allons pas appliquer telles quelles les méthodes du paragraphe 4.2.2. Il nous faut définir tout au long de la trajectoire une orientation par défaut qui permet de créer un objet correct si l'utilisateur ne manipule que les positions. Cette orientation par défaut est calculée à partir du trièdre de Frénet et ne dépend que de la trajectoire. Pour rester compatible avec notre représentation des orientations, nous utiliserons une représentation sous forme de quaternions de ce repère. En un point t de la trajectoire, nous noterons ce quaternion $q_{Frnet}(t)$ (les détails du calcul de ce quaternion seront donnés au paragraphe 4.5). La surface engendrée par le cône généralisé, sans appliquer de rotations supplémentaires, est alors donnée par

$$CG(t,s) = T(t) + q_{Frnet}(t).S(s).q^{-1}_{Frnet}(t)$$

Le repère de Frénet permet de définir les valeurs par défaut des orientations clés et des orientations intermédiaires (figure 4.4).

La fonction additionnelle définissant la variation de l'orientation au cours de la trajectoire (par rapport à l'orientation par défaut de Frénet) est réalisée en modifiant les orientations-clés. L'interpolation de ces orientations-clés fournit une fonction donnant l'évolution des ces orientations en fonction du paramètre u. Il est important de noter que l'on n'interpole pas les orientations-clés telles qu'elles apparaissent à l'écran, mais les orientations-clés définies relativement aux repères de Frénet.

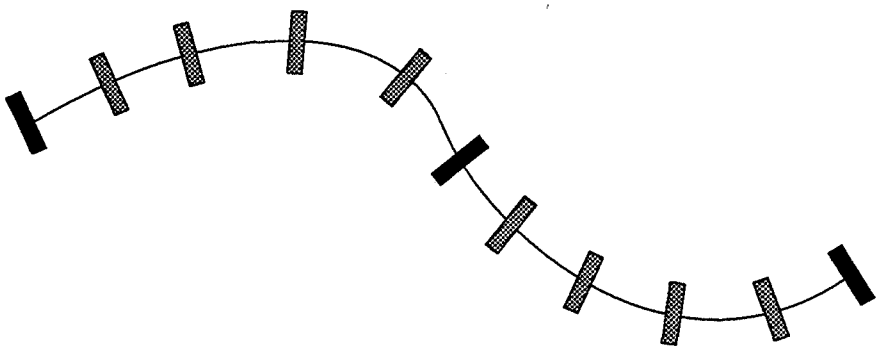


FIG. 4.4 Les orientations par défaut sont déterminées par le repère de Frénet (les orientations clés sont en noir).

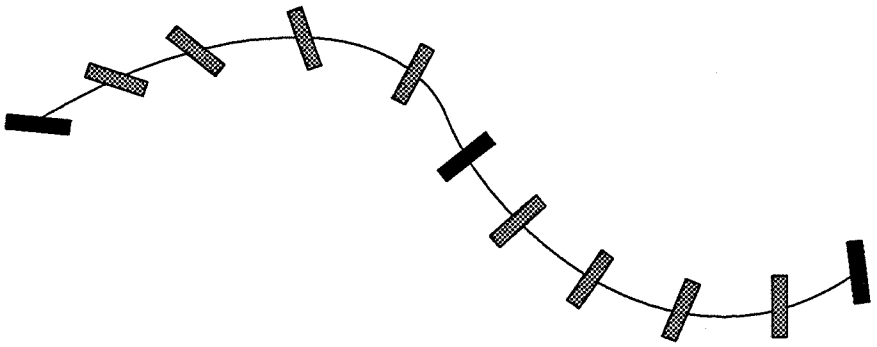


FIG. 4.5 Par rapport à la figure précédente, une rotation a été introduite sur la première et la dernière orientation-clé.

L'orientation finale q_{f_i} au point i est la composition d'une rotation q_{Frnet_i} et d'une rotation introduite par l'utilisateur q_i. Ce sont les quaternions $q_i = q_{Frnet_i}.q_{f_i}^{-1}$ que nous interpolons avec les techniques décrites dans le premier chapitre. Le résultat de cette interpolation est une trajectoire $q(t)$. Pour une valeur de t donnée, l'orientation finale de la section est donc donnée par la composée de l'orientation interpolée et de l'orientation de Frénet (figure 4.5)

$$q_f(t) = q(t).q_{Frnet}(t)$$

La surface engendrée par le cône généralisé est donc :

$$CG(t,s) = T(t) + q_f(t).S(s).q_f^{-1}(t)$$

4.5 Le repère de Frénet

4.5.1 Définition

Le repère de Frénet, lié à une trajectoire paramétrique $T(t)$, est défini par un triplet de vecteurs normés $(t(t), n(t), b(t))$ tels que :

- $t(t)$ représente la tangente à la trajectoire

$$t(t) = \frac{T'(t)}{\|T'(t)\|}$$

- $n(t)$, appelée normale principale, est un vecteur dirigé vers le centre de courbure

$$n(t) = \frac{t'(t)}{\|t'(t)\|}$$

et comme $\|T'(t)\|' = \frac{T'(t).T''(t)}{\|T'(t)\|}$, on a :

$$n(t) = \frac{T''(t)\|T'(t)\|^2 - T'(t)\|T'(t).T''(t)\|}{\|T''(t)\|T'(t)\|^2 - T'(t)\|T'(t).T''(t)\|\|}$$

- $b(t)$, appelée binormale, est le vecteur orthogonal au deux précédents

$$b(t) = t(t) \wedge n(t)$$

Lors du balayage, les deux axes par rapport auxquels est définie la section sont mis en correspondance avec la normale et la binormale.

4.5.2 Singularités du repère de Frénet

Lors de l'utilisation du repère de Frénet, nous pouvons être confrontés à trois types de problèmes [SW92] :

- La normale n'est pas définie quand la courbure est nulle.
- La normale peut tourner de façon incontrôlée autour de la trajectoire.
- La normale s'inverse aux points d'inflexion.

Le premier cas est généralement dû à une portion linéaire de la trajectoire (rayon de courbure infini). Pour l'interpolation que nous avons choisie, cela se produit quand le vecteur joignant deux points de contrôle consécutifs et les deux demi-tangentes associées sont colinéaires. Ce cas est aisément résolu en s'assurant que la normale à la fin du segment égale celle du début.

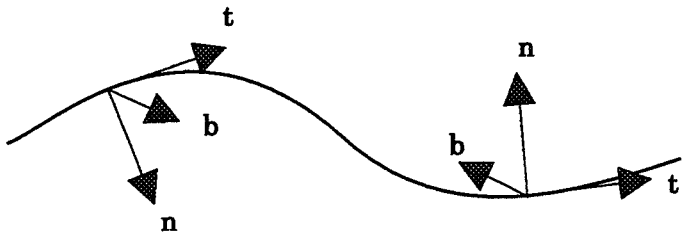


FIG. 4.6 La normale s'inverse au passage d'un point d'inflexion.

Le deuxième phénomène est entièrement dépendant de l'allure de la courbe. Il se produit quand le plan, dans lequel est défini le cercle osculateur, change fréquemment et se mesure à partir du paramètre de torsion de la trajectoire. Une extension du repère de Frénet minimisant ce type de rotation est exposé dans [Klo86].

Enfin, la singularité du point d'inflexion se traduit par une brusque inversion de la normale (figure 4.6). Dans le cas d'une trajectoire planaire, le centre de courbure passe soudainement d'un côté de la courbe à l'autre. Pour détecter un point d'inflexion, il suffit de vérifier si la binormale aux points de contrôle concernés a ou non été transformée en son opposé. En dimension trois, le problème est moins évident. Par exemple, si on effectue trois projections orthogonales d'une même courbe, l'une de ces projections peut présenter un point d'inflexion alors que les deux autres non.

Des solutions ont été proposées pour résoudre le problème des points d'inflexion. Dans [CH89], Hégron et Cousin font une recherche dichotomique des singularités sur chacun des morceaux de la courbe (ce sont des B-splines d'approximation). La subdivision n'est nécessaire que si l'angle entre deux normales consécutives est supérieur à $\frac{\pi}{2}$ radians. Quand un point d'inflexion est rencontré, la normale est inversée.

Dans [SW92], les auteurs utilisent une méthode originale consistant à calculer la normale au point clé $i+1$ à partir de la tangente en ce point clé et de la normale au point clé précédent. La normale n_{i+1} est définie comme la projection dans le plan orthogonal à t_{i+1} de la normale n_i. Dans certains cas particuliers, il sont cependant obligés de recourir à une méthode de subdivision telle que celle présentée ci-dessus. Une autre approche, exposée dans [Ple89], détermine chaque orientation à partir de la précédente. Pour passer de l'une à l'autre, on effectue la rotation minimale autour d'un axe perpendiculaire aux tangentes (au sens du repère de Frénet) des deux repères. Dans les deux cas, les orientations peuvent parfois être déconcertantes comme le montre la figure 4.7. De plus, si la trajectoire est fermée, les orientations des deux extrémités ont peu de chance de concorder.

De notre côté, l'utilisation de l'interpolation d'Hermite pour représenter la trajectoire nous a dicté la marche à suivre. Par rapport aux splines d'ordre élevé (méthode de Hégron et Cousin), le traitement est simplifié. Les points d'inflexion peuvent se situer à deux

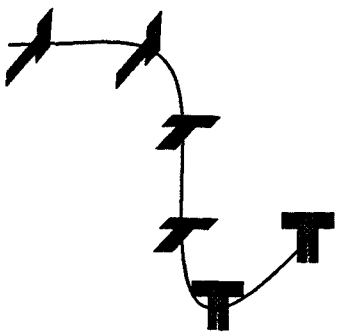


FIG. 4.7 Exemple de problème d'orientation.

niveaux :

- sur la courbe interpolant deux points P_i et P_{i+1} (figure 4.8.a). Pour détecter si un point d'inflexion existe, il faut résoudre $P_i''(u) = 0$ [Gas90], c'est à dire :

$$P_i(u) = \begin{pmatrix} 6t & 2 & 0 & 0 \end{pmatrix} M_{Hermite} \begin{pmatrix} P_i \\ R_i \\ P_{i+1} \\ G_i \end{pmatrix} = 0 \qquad (4.1)$$

  Un point d'inflexion existe si les trois équations en x, y et z résultant de (4.1) sont vérifiées par le paramètre u ($0 \leq u \leq 1$). L'équation est linéaire; au maximum, il n'y a qu'un point d'inflexion par morceau de courbe.

- A la jonction de deux morceaux de courbes $P_i(u)$ et $P_{i+1}(u)$ (figure 4.8.b). Le cas est différent du précédent puisque les normales sont définies en ce point sur les deux tronçons. Pour détecter un point d'inflexion, il suffit de regarder si la normale en $P_i(1)$ et celle en $P_{i+1}(0)$ sont ou non égales.

Si l'utilisateur introduit une discontinuité dans la direction des tangentes, la normale à la courbe sera discontinue, entrainant des effets de torsion impossibles à prendre en compte.

Ces deux traitements nous permettent de déterminer l'ensemble des valeurs de t ($0 \leq t \leq n$) qui admettent un point d'inflexion. Pour orienter correctement le repère de Frénet en un point t, il suffit de compter à partir de $t = 0$ le nombre de points d'inflexion rencontrés. Si ce nombre est pair, le trièdre est inchangé, s'il est impair, la normale et la binormale sont inversées.

4.5.3 Détermination du quaternion de Frénet

Nous allons déterminer le quaternion $q = [q_w, (q_x, q_y, q_z)]$ qui transforme le repère du monde en ce trièdre. Le vecteur $x = (1, 0, 0)$ se transforme en $t = (t_x, t_y, t_z)$, la section

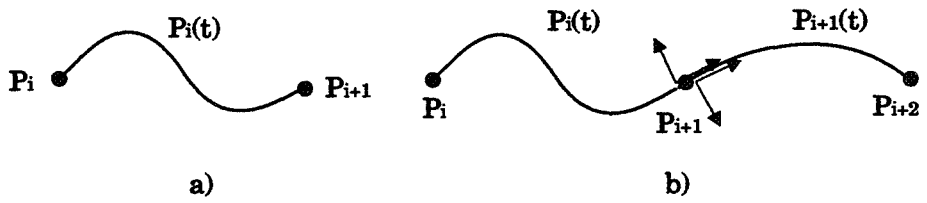


FIG. 4.8 Les deux cas d'inflexion.

étant contenue dans le plan yz. En appliquant la formule de transformation d'un vecteur par un quaternion, on a:

$$t_x = q_x^2 + q_w^2 - q_z^2 - q_y^2 \tag{4.2}$$
$$t_y = 2q_xq_y + 2q_wq_z \tag{4.3}$$
$$t_z = 2q_xq_z - 2q_yq_w \tag{4.4}$$

De même, le vecteur $y = (0,1,0)$ se transforme en $n = (n_x, n_y, n_z)$ et le vecteur $z = (0,0,1)$ se transforme en $b = (b_x, b_y, b_z)$:

$$n_x = 2q_yq_x - 2q_wq_z \tag{4.5}$$
$$n_y = q_y^2 + q_w^2 - q_x^2 - q_z^2 \tag{4.6}$$
$$n_z = 2q_wq_x + 2q_zq_y \tag{4.7}$$
$$b_x = 2q_zq_x + 2q_yq_w \tag{4.8}$$
$$b_y = 2q_zq_y - 2q_xq_w \tag{4.9}$$
$$b_z = q_z^2 + q_w^2 - q_y^2 - q_x^2 \tag{4.10}$$

A partir de là, on peut déterminer q_w en additionnant les équations (4.2), (4.6) et (4.10):

$$\begin{aligned} t_x + n_y + b_z &= 3q_w^2 - q_x^2 - q_y^2 - q_z^2 \\ &= 4q_w^2 - \left(q_w^2 + q_x^2 + q_y^2 + q_z^2\right) \end{aligned}$$

Comme nous n'utilisons que des quaternions de norme 1, on a:

$$q_w^2 = \frac{1}{4}(t_x + n_y + b_z + 1)$$

Si q_w^2 est positif, la partie réelle du quaternion est égale à:

$$q_w = \frac{1}{2}\sqrt{(t_x + n_y + b_z + 1)}$$

et on utilise les équations restantes de la façon suivante:

$$(4.8) - (4.4) \qquad b_x - t_z = 4q_yq_w$$
$$(4.3) - (4.5) \qquad t_y - n_x = 4q_zq_w$$
$$(4.7) - (4.9) \qquad n_z - b_y = 4q_xq_w$$

ce qui nous donne la composante imaginaire de q:

$$\begin{aligned} q_x &= \frac{1}{4q_w}(n_z - b_y) \\ q_y &= \frac{1}{4q_w}(b_x - t_z) \\ q_z &= \frac{1}{4q_w}(t_y - n_x) \end{aligned}$$

Dans le cas contraire, on fixe $q_w = 0$ et on additionne l'équation (4.6) à l'équation (4.10) ce qui donne :

$$\begin{aligned} n_y + b_z &= 2q_w^2 - 2q_x^2 \\ q_x^2 &= \frac{1}{2}(n_y + b_z) \end{aligned}$$

Si q_x^2 est positif, $q_x = \sqrt{\frac{1}{2}(n_y + b_z)}$, $q_y = \frac{n_x}{2q_x}$ et $q_z = \frac{b_x}{2q_x}$. Dans le cas contraire, on fixe $q_x = 0$ et l'équation (4.10) nous donne :

$$\begin{aligned} b_z &= q_z^2 - q_y^2 \\ &= 1 - 2q_y^2 \end{aligned}$$

d'où

$$q_y^2 = \frac{1}{2}(1 - b_z)$$

Si $q_y^2 > 0$, $q_y = \sqrt{\frac{1}{2}(1 - b_z)}$ et l'équation (4.7) nous donne $q_z = \frac{n_x}{2q_y}$. Dans le cas contraire, on fixe $q_y = 0$ et le fait que q soit un quaternion unitaire impose finalement $q_z = 1$.

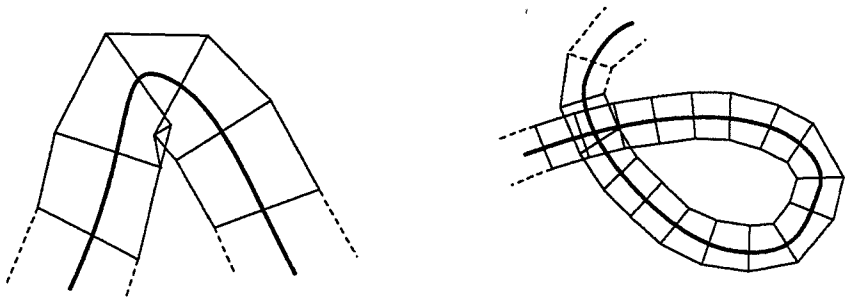
On pourra retrouver une synthèse de ces résultats sous forme de fonction en C dans l'annexe A.

4.6 Construction d'un cylindre généralisé

4.6.1 Enveloppe polygonale

Certains auteurs [CH89] représentent un cylindre généralisé sous la forme d'une surface spline ce qui a l'avantage principal de permettre de reconstruire le solide selon le degré de précision voulu. Ici, nous décrivons comment est évaluée la frontière discrétisée.

L'échantillonnage de la trajectoire se fait, soit en utilisant un pas constant sur les valeurs prises par le paramètre t, soit en la subdivisant en fonction d'un critère de linéarité. Pour déterminer l'échantillonnage des sections le long de la courbe, nous utilisons l'algorithme de reparamétrisation décrit dans le chapitre précédent. Ici, l'intérêt de cette



a) b)

FIG. 4.9 Les deux cas d'auto-intersection.

paramétrisation est encore renforcé. Il est en effet primordial de répartir de nombreux échantillons dans les zones à forte courbure; au contraire, les portions quasi-rectilignes ne requièrent qu'une facettisation minimale.

Le résultat de cette paramétrisation automatique est un ensemble de valeurs t_i dont la répartition tient compte de la courbure de la trajectoire. En prenant n échantillons sur la trajectoire et m points sur la section (représentée par un polygone), les points

$$\{T(t_i) + q_{Frnet}(t_i)S(j)q^{-1}_{Frnet}(t_i), 1 \leq i \leq n, 1 \leq j \leq m\}$$

constituent la grille recouvrant le cylindre généralisé.

4.6.2 Cas d'auto-intersection

Lors de la génération de l'enveloppe polygonale de l'objet, deux cas d'auto-intersection peuvent survenir:

- Quand le rayon de courbure de la trajectoire est inférieur au rayon de la section, deux sections consécutives peuvent s'intersecter (figure 4.9.a).
- Le profil intersecte une partie du cylindre déjà construite. Cela se produit quand, par exemple, la trajectoire décrit une boucle (figure 4.9.b).

Si l'on ne désire qu'une représentation par frontière de l'objet – où les notions d'intérieur et d'extérieur n'existent pas – les phénomènes d'auto-intersection ne posent pas de problème majeur. Un algorithme de type balayage de ligne affichera, pour un pixel donné, la première facette traversée.

Quand la représentation volumique de l'objet est nécessaire (application de différences sur l'objet dans un modèle CSG), nous décomposons le cylindre en un ensemble d'objets plus petits tels que ceux-ci soient réguliers, i.e. ils ne présentent pas les inconvénients énumérés ci-dessus. Ensuite, il suffit de dire que le cylindre final est la réunion – au sens opérations booléennes des arbres de construction – de l'ensemble de ces objets réguliers. La construction de l'arbre de construction se fait itérativement en considérant, l'un après

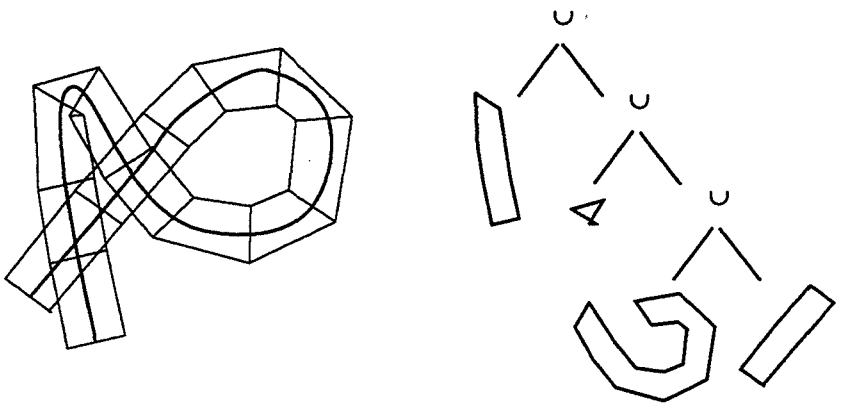


FIG. 4.10 Création de l'arbre CSG représentant le cône.

l'autre, tous les tronçons constituant le cylindre généralisé. Un tronçon est le volume balayé entre deux échantillons consécutifs de la section.

Suivant le type d'auto-intersection, la décomposition se fait de deux manières différentes. Dans le premier des cas, les deux côtés limitant le tronçon s'intersectent. Nous suggérons de construire l'enveloppe convexe à partir de l'ensemble des points appartenant aux deux sections incriminées (cette partie n'a pas été implantée).

Dans le deuxième cas, une section du cylindre intersecte une partie du cylindre déjà construite. On sépare alors la partie qui n'est pas encore construite (et qui contient la section en cause) de la portion du cylindre déjà calculée. Pour cela, on ajoute un nouveau nœud dans l'arbre booléen contenant l'objet en construction. Le cylindre est donc la réunion de la partie construite et de la partie à construire. Notez qu'il n'est plus nécessaire de détecter les intersections entre les deux parties de l'arbre; si intersection il y a, elle sera résolue lors de l'évaluation de l'arbre. L'arbre booléen résultant est de type peigne. Chaque nœud de l'arbre final indique un cas d'auto-intersection. La figure 4.10 illustre la transformation d'un cylindre généralisé en arbre de construction.

Suivant la façon dont va être utilisé l'arbre booléen définissant le cône, deux solutions s'offrent à nous. Si l'on ne désire que visualiser le résultat, on peut le faire directement en utilisant un algorithme sachant traiter les arbres CSG (tracé de rayon ou Atherton). Par contre, si l'on désire obtenir la description volumique de l'objet résultant, nous avons à notre disposition un algorithme permettant de résoudre explicitement les opérations booléennes [BMP93]. Le résultat sera une primitive pouvant à son tour être intégrée dans un modèle par arbre de construction.

4.7 Exemples

La figure 4.11 présente un cylindre généralisé. Seule l'orientation de Frénet définit les orientations. Sur la figure 4.12, une rotation supplémetaire a été ajoutée autour de la tangente. L'effet de torsion est obtenu en perturbant les orientations clés. Une rotation autour d'un axe passant par la section a permis d'obtenir l'illustration 4.13. Enfin, la figure 4.14 donne un exemple d'auto-intersection.

4.8 Conclusion

Nous avons présenté dans ce chapitre une méthode permettant la modélisation interactive des cylindres généralisés. Notre approche se distingue des précédentes par la gestion de toutes les composantes du cylindre généraliséffi dans la même fenêtre et l'extension de la notion de rotation. Les développements concernant le contrôle des rotations nous ont permis d'associer à la définition du cylindre généraliséffi une trajectoire en orientation disposant de trois degrés de liberté. Par contre, nous avons laissé de côté des règles de balayage supplémentaires. L'ajout de règles décrivant l'évolution de la taille de la section le long de la trajectoire pourrait facilement être réalisé. Il faudrait néanmoins modifier le formalisme basé sur les quaternions. Celui-ci ne permet pas en effet d'intégrer directement d'autres règles de balayage basées sur des matrices. Heureusement, la conversion d'un quaternion en une matrice de rotation est bien connue.

D'autre part, l'algorithme de paramétrisation présenté dans le chapitre précédent est particulièrement bien adapté à l'échantillonnage des sections lors de la construction du cylindre généralisé. Enfin, la méthode de construction du cylindre convertit celui-ci en un arbre CSG dont la résolution aboutit à un objet homogène et cohérent.

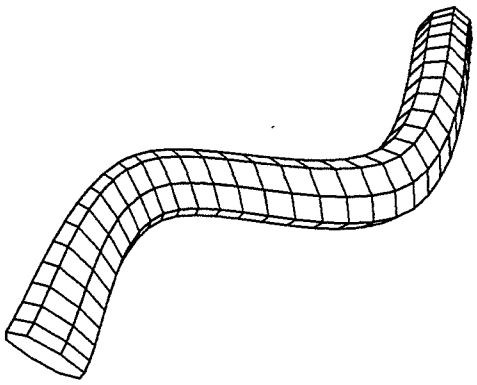


FIG. 4.11 Extrusion d'un cercle. Les orientations sont déterminées par le repère de Frénet.

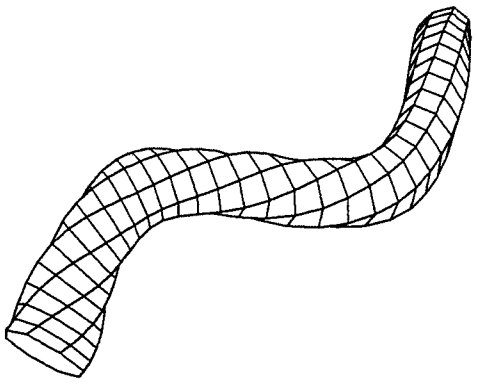


FIG. 4.12 Rotation autour de la trajectoire.

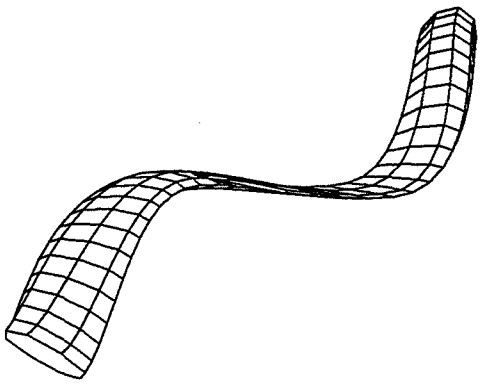


FIG. 4.13 Rotation par rapport à un axe perpendiculaire à la trajectoire.

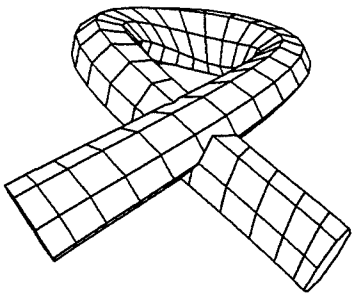


FIG. 4.14 Objet auto-intersectant.

Deuxième partie

Interpolation par des méthodes d'optimisation

Introduction

Dans la première partie de ce rapport, nous avons proposé quelques techniques permettant de faciliter le contrôle de l'animateur sur la description des animations. Cependant, malgré toutes les améliorations possibles, l'animation basée sur la notion de paramètres-clés ne permet pas de créer facilement des mouvements réalistes. En particulier, dès que l'objet à animer devient complexe, il devient pratiquement impossible de générer une animation réaliste, et ce, malgré le talent de l'animateur. La cause en est simple : les mouvements réels sont soumis à des lois complexes.

D'un autre côté, l'animation basée sur des modèles causaux, c'est à dire qui tiennent compte des causes qui engendrent le mouvement, a prouvé ses capacités à engendrer simplement des mouvements imitant ceux de la réalité. De nombreux papiers publiés ces dernières années démontrent l'intérêt de la dynamique en animation par ordinateur [HAD88, Ver89, Gas89, AG85, Wil87, LC86, GG92]. Globalement, l'objectif de la majorité de ces papiers est le même : fournir une méthodologie permettant d'obtenir les équations mécaniques du mouvement. On peut ainsi calculer le mouvement d'un solide complexe à partir des lois de la dynamique, des forces qui entrent en jeu et des positions initiales de l'objet. En spécifiant un minimum de données, il est possible de créer des mouvements complexes. L'inconvénient est qu'une fois les paramètres initiaux fournis, les objets sont livrés à eux-mêmes, cantonnant ainsi l'animateur à un rôle de spectateur. Le contrôle de l'animation est difficile et demande à l'animateur de deviner les forces permettant à l'objet d'atteindre une position déterminée. Par exemple, le mouvement d'une balle lancée en l'air est facile à calculer. Trouver les forces qui vont faire rebondir la balle en un endroit spécifique est bien plus compliqué.

Certaines approches ont permis d'augmenter les possibilités de contrôle sur des animations basées sur la dynamique; soit par des méthodes de dynamique inverse [IC87], soit par l'intermédiaire de contraintes cinématiques [BB88][WW90][BC89]. En fait, elles permettent à l'utilisateur de spécifier la trajectoire de certains éléments de l'animation. Le système d'animation calcule alors les forces qui permettent de respecter ces trajectoires et en déduit éventuellement le mouvement des éléments libres. Par contre, il n'est pas possible de trouver les forces – et par conséquent la trajectoire – permettant de conduire un objet d'un endroit initial à un état final spécifié.

Des approches apparues récemment répondent à ce type de problème. Elles permettent de faire coopérer les modèles descriptifs et les modèles causaux. Elles profitent des avantages des deux méthodes. L'animateur garde le contrôle du mouvement en définissant

des positions-clés mais le mouvement est calculé à partir des lois de la dynamique. Ces solutions permettent de réaliser des interpolations qui vérifient les lois de la dynamique.

Les équations du mouvement sont des équations différentielles d'ordre deux (faisant intervenir des dérivées secondes) exprimant les accélérations linéaires ou angulaires en fonction des paramètres de vitesse et de position. La simulation, ou mouvement en dynamique directe, est un problème différentiel avec valeurs initiales. Ici au contraire, il s'agit de résoudre un problème différentiel avec valeurs aux deux bornes. C'est un problème bien plus complexe.

Il existe une infinité de forces – donc de trajectoires – permettant d'aller des positions initiales aux positions finales, même en respectant les lois de la dynamique. Le point commun des méthodes évoquées est d'utiliser un critère d'optimisation pour choisir, parmi toutes les trajectoires possibles, celle qui fournira le "meilleur" mouvement. En général, ce critère est choisi de manière à minimiser l'énergie dépensée pendant l'animation, ce qui semble correspondre à un principe naturel. Jusqu'à présent, trois solutions différentes ont été proposées pour résoudre ce problème de minimisation.

La première, due à Brotman et Netravali [BN88], utilise la théorie du contrôle optimal. Ils se restreignent à des mouvements dépendant de lois linéaires du mouvement. Celles-ci sont traitées sous leur forme continue.

Witkin et Kass [WK88] utilisent pour leur part les méthodes de l'optimisation. C'est un problème de minimisation de variables sous contraintes. Pour ce faire, les équations différentielles sont discrétisées et intégrées au problème en tant que contraintes.

Enfin, Van de Panne, Fiume et Vranesic [dPFV90] utilisent une technique d'optimisation combinatoire appelée programmation dynamique. Là aussi, les équations sont discrétisées.

Ces trois approches, identiques dans leur principe, mais fondées chacunes sur une technique d'optimisation différente, seront détaillées et comparées dans les pages qui viennent. Par ailleurs, il nous a semblé intéressant d'explorer un domaine pour l'instant délaissé: l'utilisation de la théorie du contrôle optimal pour les équations non linéaires. Contrairement à la méthode de Brotman et Netravali, cela permet de prendre en compte n'importe quel type d'équation. Ces derniers ne présentent d'ailleurs que des applications simples.

Cette deuxième partie est composée de deux chapitres. Le premier exposera le choix du formalisme mécanique qui a été retenu. Celui-ci a été dicté par les contraintes qu'imposent les techniques du contrôle optimal:

- Les équations du mouvement doivent exprimer les dérivées secondes (accélérations angulaires) en fonction des dérivées des paramètres et des paramètres eux-même. Si q désigne l'ensemble des variables mécaniques du systèmes et u les forces agissant sur ces variables, les équations doivent être sous la forme:

$$\ddot{q} = f(\dot{q}, q, u, t)$$

- Il est nécessaire de pouvoir calculer les dérivées partielles des équations du mouvement, par rapport à q et par rapport à u. Le formalisme mécanique doit donc permettre de construire des équations du mouvement sous forme symbolique.

Accessoirement, ces équations – utilisées sous leur forme numérique et avec des valeurs initiales – ont permis de générer des animations en dynamique directe.

Dans le deuxième chapitre, nous parlerons des différentes techniques d'optimisation, et particulièrement de celle que nous proposons. Les références théoriques de chaque méthode seront détaillées pour mettre en évidence leurs particularités et différences respectives. Pour chacune d'elle, un exemple illustrera l'application de la méthode théorique pour une animation simple.

La conclusion sera l'occasion d'un bilan comparatif.

Chapitre 5

Equations du mouvement

5.1 Les différents formalismes

Depuis les années 80, plusieurs formalismes ont été proposés pour exprimer les équations du mouvement d'un système articulé soumis aux lois de la dynamique [DGV91]. On distingue essentiellement les approches suivantes :

- Wilhelms et Barsky [WB85] proposent une méthode qui a été finalement abandonnée au profit de la suivante car la résolution du mouvement est très coûteuse.
- Armstrong et Green [AG85] [AGL87] animent un solide ayant une structure arborescente. Les équations sont d'abord écrites pour chaque élément de l'arbre puis propagées aux nœuds voisins par un parcours récursif de l'arbre. Ils utilisent le formalisme mécanique de Newton-Euler.
- Zeltzer et McKenna [MZ90], Zeltzer et Schröder [SZ90] utilisent eux aussi une approche récursive. Le formalisme mécanique est basé sur celui de Featherstone [Fea83]
- Isaac et Cohen [IC87] appliquent la méthode des travaux virtuels décrit par exemple dans [Wit77]. Les équations finales sont sous la forme $A\ddot{q} = B$, ce qui leur permet de combiner simulation en dynamique directe et dynamique inverse.
- Arnaldi, Dumont et Hégron [DAH89] appliquent les équations de Lagrange.

Les modèles récursifs ne sont pas adaptés à la construction symbolique des équations du mouvement. Les accélérations d'un corps dépendent de celles du père. La résolution du mouvement se fait en effectuant un premier parcours de l'arbre, des feuilles vers la racine, puis un second, en redescendant de la racine vers les feuilles.

Les méthodes basées sur les équations de Lagrange permettent d'obtenir des équations de la forme $f(\ddot{q}, \dot{q}, q, t) = 0$. Elles n'expriment pas directement les accélérations des paramètres en fonction des vitesses et des paramètres eux-mêmes. D'autre part, des variables auxiliaires dépendantes des paramètres généraux ajoutent des contraintes au système. Par

exemple, dans l'article de Arnaldi, Dumont et Hégron, la position de l'extrémité d'une chaîne articulée est exprimée en fonction des variables articulaires. Cela permet de contrôler le mouvement de l'élément terminal du bras sans avoir à se soucier des valeurs que prennent les angles des différentes liaisons. Ces contraintes sont intégrées, soit avec les multiplicateurs de Lagrange, ce qui introduit des inconnues et des équations supplémentaires par rapport aux équations initiales, soit à l'aide de la méthode des pénalités.

La fin de ce paragraphe expose les raisons ayant conduit au choix du formalisme mécanique pour l'algorithme de contrôle optimal.

Dans le cas des méthodes à base d'algorithme d'optimisation, les contraintes dont nous parlions plus haut sont difficiles à prendre en compte. Si l'on utilise les multiplicateurs de Lagrange, elles requièrent une méthodologie plus complexe (résolution avec contraintes). De son côté, l'utilisation du terme de pénalité rend la résolution instable : si le critère à minimiser dans un algorithme d'optimisation contient un terme de pénalité, les contours à valeur constante du critère décrivent des ellipses très allongées. Un dernier inconvénient est que l'utilisateur doit fournir lui-même l'expression de l'énergie cinétique et des travaux [DAH89, WK88], ce qui n'est pas le cas avec la solution que nous avons retenue.

Nous avons choisi d'utiliser le formalisme de Wittenburg. Une partie de ce formalisme a déjà été utilisée par Issac et Cohen, mais avec une optique différente puisqu'il s'agissait de résoudre des problèmes de dynamique inverse [IC88]. Le principe de la méthode repose sur l'utilisation d'une représentation matricielle de la connexité du système. Des opérations de calcul matriciel permettent d'obtenir les équations du mouvement sous la forme dont nous parlions plus haut ($A\ddot{q} = B$). Ici, A est la matrice généralisée des masses et B est un vecteur contenant les forces appliquées. Par inversion de A, le système peut-être mis sous une forme plus appropriée $\ddot{q} = f(\dot{q}, q, t)$. Les contraintes de liaison sont intégrées dans ces équations que nous allons maintenant détailler.

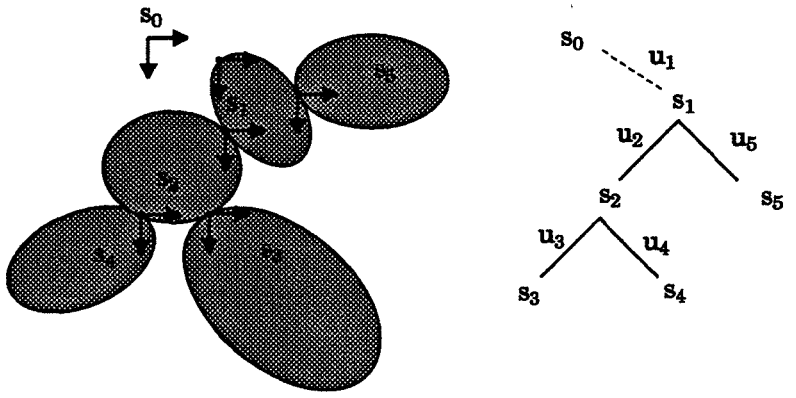
5.2 Méthode de Wittenburg

Nous nous limiterons ici aux systèmes dotés d'une structure arborescente tels que ceux décrits précédemment dans la deuxième partie du chapitre 2. D'autre part, les seules liaisons permises sont des liaisons rotoïdes. Notons néanmoins que le formalisme de Wittenburg autorise les liaisons de tout type ainsi que des cycles dans la structure du système.

5.2.1 Structure du système

Le système mécanique est composé de n corps rigides que nous noterons s_i $(1 \leq i \leq n)$ plus un corps fictif s_0 auquel est reliée la racine de l'arbre. s_0, doté lui de six degrés de liberté, peut éventuellement être soumis à une accélération linéaire et angulaire. Des liaisons notées u_i permettent de relier les différents corps entre eux. Il y autant de liaisons que de solides (en comptant la liaison entre s_0 et la racine de l'arbre).

Le système peut être représenté sous forme de graphe (en fait un arbre) où les sommets sont les corps et les arêtes, les liaisons. Ce graphe doit être orienté; c'est important quand



a) b)

FIG. 5.1 Exemple de solide rigide articulé (a) et son arbre de connexité associé (b)

on décrira le mouvement relatif de chaque corps par rapport à ses voisins. Les forces internes de liaison entre deux corps sont en effet de signes opposés.

Les sommets du graphe sont numérotés de telle sorte que tout sommet s_i se trouvant sur le chemin entre s_0 et un sommet s_j vérifie $i < j$. Pour cela, il suffit de numéroter les feuilles de l'arbre avec les indices n, $n-1$, etc...La méthode est ensuite appliquée avec les numéros restant sur l'arbre dont les sommets déjà étiquetés ont été enlevés. Par la suite, on utilisera le raccourci $s_i < s_j$ pour indiquer que le solide s_i se situe sur le chemin allant de s_0 à s_j. La figure (5.1) montre l'exemple d'un système articulé composé de cinq solides élémentaires. Chacun d'eux est doté de son propre repère local fixé sur la liaison.

La structure du système définit de manière unique deux fonctions $i^+(a)$ et $i^-(a)$ qui établissent les relations entre la liaison a et les solides qu'elle relie. $i^+(a)$ est l'indice du nœud de l'arbre d'où provient l'arête a et $i^-(a)$ est l'indice du nœud de l'arbre vers où pointe a. Dans la structure arborescente, $i^+(a)$ et le père de $i^-(a)$ à travers la liaison a. Sur l'exemple donné, nous avons:

$$\begin{aligned} i^+(1) &= 0, \\ i^+(2) &= 1, \\ i^+(3) &= 2, \\ i^+(4) &= 2, \\ i^+(5) &= 1 \end{aligned}$$

Les liaisons sont numérotées de telle sorte que $i^-(a) = a$. Chaque liaison possède donc l'indice du sommet d'où elle est issue.

A partir de ces deux fonctions, la matrice d'incidence S peut être construite. Elle est

définie de la façon suivante :

$$S_{ia} = \begin{cases} +1 & \text{si } i = i^+(a) \\ -1 & \text{si } i = i^-(a) \\ 0 & \text{sinon} \end{cases}$$

Si on considère à nouveau notre système à cinq corps, la matrice S s'écrit :

$$S = \begin{pmatrix} -1 & 1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

Une autre matrice remarquable est la matrice T définie par :

$$T_{ai} = \begin{cases} +1 & \text{si l'arc } u_a \text{ est sur le chemin entre } s_0 \text{ et } s_i \text{ et va vers } s_0 \\ -1 & \text{si l'arc } u_a \text{ est sur le chemin entre } s_0 \text{ et } s_i \text{ et vient de } s_0 \\ 0 & \text{sinon} \end{cases}$$

Ces deux matrices sont liées entre elles par une relation importante qui permettra de simplifier les calculs par la suite :

$$TS = ST = Id$$

On en trouvera la démonstration dans [Wit77].

5.2.2 Equations pour un corps s_i

Dans ce paragraphe, les équations du mouvement pour un seul solide s_i isolé sont données. Chaque corps s_i est caractérisé par ses paramètres de masse et d'inertie[1] :

- sa masse m_i,
- sa matrice d'inertie J_i,
- son centre de masse G_i,
- une liste de vecteurs c_{ia} partant de G_i et dirigés vers l'ensemble des liaisons que possède s_i. Ces vecteurs représentent les bras de levier pour les forces de liaison agissant sur s_i (figure 5.2).

D'autre part, s_i est soumis à un ensemble de forces et de couples externes supposés connus (la gravité par exemple) ainsi qu'à des forces et des couples internes assurant le respect des liaisons :

- F_i, la somme des forces extérieures appliquées en G_i.

[1] Comme dans le paragraphe (3.5.1), ils sont approximés en utilisant la boîte englobante du corps.

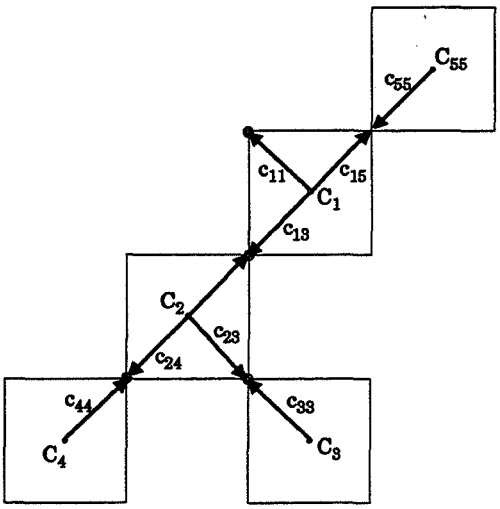


FIG. 5.2 Répartition des bras de levier du système

- M_i, la somme des couples extérieurs par rapport à G_i.

- Sur chaque liaison a, une force de contrainte X_a^C assure le respect de la liaison. Y_a est la résultante des couples internes appliqués à la liaison a (cela peut être par exemple le couple de rappel engendré par un ressort situé sur la liaison). Par convention, la liaison a engendre une force interne $+X^C(a)$ sur le corps $i^+(a)$ et une force $-X^C(a)$ sur le corps $i^-(a)$. La même convention est utilisée pour les couples Y_a. L'ensemble des forces de liaison qui agissent sur s_i s'écrit donc $\sum_{a=1}^{n} S_{ia} X_a^C$. De même, le moment résultant, dû aux actions de liaison, appliqué à s_i est $\sum_{a=1}^{n} S_{ia} (c_{ia} \wedge X_a^C + Y_a)$. On voit ici l'importance de la matrice S qui élimine implicitement de ces expressions les termes où interviennent des corps qui ne sont pas reliés directement à s_i.

Concernant le mouvement de translation, la loi de Newton appliquée au corps s_i a donc la forme:

$$m_i \ddot{r}_i = F_i + \sum_{a=1}^{n} S_{ia} X_a^C$$

où r_i désigne les coordonnées du barycentre G_i dans le repère de référence. Concernant le mouvement de rotation, l'expression est la suivante:

$$\dot{L}_i = M_i + \sum_{a=1}^{n} S_{ia} (c_i a \wedge X_a^C + Y_a)$$

où L_i désigne le moment angulaire du corps s_i par rapport à G_i.

Sous forme matricielle [2] et en appliquant les deux lois précédentes à tous les membres du système articulé, nous avons :

$$m\ddot{r} = F + SX^C \tag{5.1}$$

$$\dot{L} = M + C \wedge X^C + SY \tag{5.2}$$

m est la matrice diagonale contenant les masses; r, F, X^C, $\dot{L}$, Y et M sont des vecteurs colonnes; C est la matrice contenant les éléments $C_{ia} = S_{ia}c_{ia}$.

Il n'y a pas intérêt à ce que X^C figure dans les équations du mouvement. Pour l'éliminer, on utilise le fait que $TS = Id$ en prémultipliant l'équation (5.1) par T :

$$X^C = T(m\ddot{r} - F)$$

et en substituant cette expression dans l'équation (5.2) on obtient finalement :

$$\dot{L} - CT \wedge (m\ddot{r} - F) = M + SY \tag{5.3}$$

Cette équation détermine un ensemble de $3n$ équations scalaires qui correspondent au mouvement de chaque degré de liberté. Avant de pouvoir s'en servir, il est nécessaire de définir les relations entre les accélérations de translation intervenant dans les termes r et les accélérations angulaires de L, ou indirectement entre les positions des centres d'inertie et les orientations des repères locaux. Par exemple, la position r_i du barycentre de s_i dépend de la position de s_0 ainsi que d'une suite de vecteurs fixés dans les repères locaux des différents corps se trouvant entre s_i et s_0. Ainsi, l'accélération $\ddot{r}_i$ (dans le repère de référence) dépend de l'accélération de s_0 ainsi que d'un ensemble de termes où interviennent les vitesses et accélérations angulaires des corps se trouvant sur le chemin vers s_0.

Introduisons maintenant les vecteurs d_{ij}, fixés au corps i, qui représentent les éléments de la matrice CT :

$$d_{ij} = (CT)_{ij} = \sum_{a=1}^{n} T_{aj} S_{ia} c_{ia}$$

[2] La plupart des matrices (exception faite des matrices de connexité S et T) sont des matrices dont les éléments sont des vecteurs. Nous n'avons pas introduit de notation particulière pour les opérations faisant intervenir des matrices vectorielles. Dans le cas du produit matriciel, il suffit d'appliquer les règles habituelles tout en tenant compte du type des matrices impliquées.

Considérons par exemple le produit $C = AB$. Chaque élément c_{ij} de C est obtenu par la somme $c_{ij} = \sum_k a_{ik} \odot b_{kj}$, l'opérateur $\odot$ étant déduit du type des matrices concernées :

- Si A et B sont des matrices scalaires, $\odot$ désigne la multiplication usuelle.
- Si A est une matrice scalaire et B une matrice vectorielle, $\odot$ désigne la multiplication d'un scalaire par un vecteur et C est une matrice vectorielle.
- Si A et B sont des matrices vectorielles, $\odot$ désigne le produit scalaire et C est une matrice scalaire.

Enfin, l'expression $C = A \wedge B$ représente le produit vectoriel de deux matrices constituées de vecteurs. La matrice C est une matrice vectorielle dont les termes sont donnés par :

$$c_{ij} = \sum_k a_{ik} \wedge b_{kj}$$

d_{ij} représente, dans le repère du monde, le vecteur qui relie les liaisons joignant s_j à s_0. Quand s_j n'est pas sur le chemin entre s_i et s_0, $d_{ij} = 0$; quand $s_i = s_j$, d_{ij} relie G_i au point de liaison vers s_0. Ces résultats sont la conséquence des propriétés particulières des matrices S et T (voir [Wit77]).

A partir de là, r_i peut être calculé à partir d'une somme de vecteurs d_{ij} de la façon suivante :

$$r_i = r_0(t) - \sum_{j=1}^{n} d_{ji}$$

Toujours avec le même système articulé (figure 5.3), la position de G_3, par exemple, est la somme : $r_3 = r_0 - d_{13} - d_{23} - d_{33}$, d_{43} et d_{53} étant nuls.

Sous forme matricielle, en notant 1_n le vecteur colonne dont tous les éléments sont des 1, la relation précédente s'écrit aussi :

$$r = r_0 - (CT)^T 1_n$$

En subsituant r par $r_0 - (CT)^T 1_n$ dans l'équation (5.3), on obtient une équation où n'interviennent plus les accélérations r :

$$\dot{L} - (CT) \wedge \left(m(r_0 - (\ddot{C}T)^T 1_n) - F \right) = M + SY$$

$$\dot{L} - (CT) \wedge (m\ddot{r}_0 - F) + (CT) \wedge (\ddot{C}T)^T 1_n = M + SY \qquad (5.4)$$

Les éléments de cette équation où interviennent des accélérations angulaires sont les termes g_{ij} de la matrice $(CT) \wedge (\ddot{C}T)^T$ dont les éléments vérifient :

$$g_{ij} = \sum_{k=1}^{n} m_k d_{ik} \wedge \ddot{d}_{jk}$$

Sans détailler, suivant les relations respectives entre s_i et s_j, nous avons :

$$g_{ij} = \begin{cases} \sum_{k=1}^{n} m_k d_{ik} \wedge \ddot{d}_{ik} & \text{si} \quad S_i = S_j \\ d_{ij} \wedge \sum_{k=1}^{n} m_k \ddot{d}_{jk} & \text{si} \quad S_i < S_j \\ \sum_{k=1}^{n} m_k d_{ik} \wedge \ddot{d}_{ji} & \text{si} \quad S_i > S_j \\ 0 & \text{sinon} \end{cases}$$

C'est ici que la notion de corps augmenté intervient pour simplifier ces expressions. A chaque corps s_i du système, un corps augmenté est associé. A chaque point de liaison c_{ia} que possède s_i est attachée une masse égale à la somme des masses des corps connectés directement ou indirectement à s_i par l'intermédiaire de la liaison a. Tous les corps augmentés ont donc la même masse, égale à la masse totale du système. Si l'on reprend notre exemple, le corps augmenté associé à s_2 est obtenu en attachant une masse m_4 au levier c_{24}, une masse m_3 au levier c_{23} et une masse $m_1 + m_4$ au levier c_{21} (figure 5.3.b).

Le centre de masse du corps augmenté s_i est noté B_i. Les vecteurs b_{ij} sont alors définis. Quand $j = i$, b_{ii} est le vecteur joignant B_i à G_i. Quand $j \neq i$, b_{ij} relie B_i à l'extrémité de la liaison a qui mène directement ou indirectement vers j.

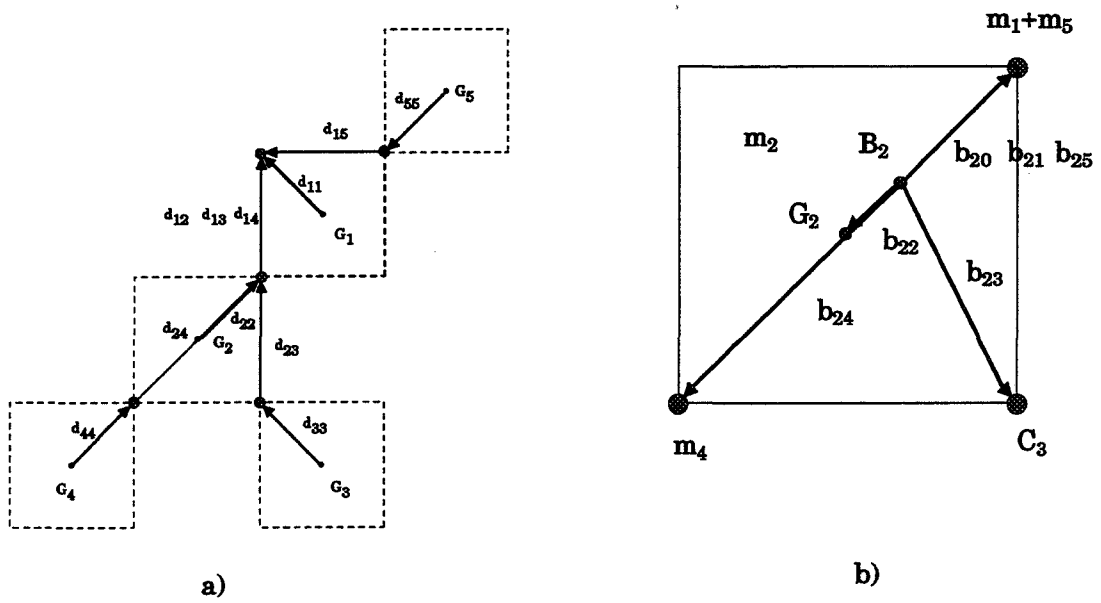


FIG. 5.3 Les vecteurs d_{ij}

Les vecteurs d_{ij} et b_{ij} sont liés par la relation $d_{ij} = b_{i0} - b_{ij}$ qui permet de simplifier l'expression des g_{ij} :

$$g_{ij} = \begin{cases} \sum_{k=1}^{n} m_k d_{ik} \wedge \ddot{d}_{ik} & \text{si} \quad S_i = S_j \\ M d_{ij} \wedge \ddot{b}_{j0} & \text{si} \quad S_i < S_j \\ M b_{i0} \wedge \ddot{d}_{ij} & \text{si} \quad S_i > S_j \\ 0 & \text{sinon} \end{cases}$$

En remplaçant ces valeurs dans l'expression générale (équation 5.4), on obtient une expression plus longue mais bien plus simple à évaluer. De nombreuses sommations ont disparu. Ecrite cette fois pour un corps i, cela donne :

$$\dot{L}_i + \sum_{j=1}^{n} g_{ij} - \sum_{j=1}^{n} (d_{ij} \wedge m\ddot{r}_0 - F_j) = M_i + \sum_{a=1}^{n} S_{ia} Y_a$$

Si l'on note par $j : S_i < S_j$ l'ensemble des corps S_j dont le chemin vers la racine passe par S_i, on obtient :

$$\dot{L}_i + \sum_{k=1}^{n} m_k d_{ik} \wedge \ddot{d}_{ik} + M \left(\sum_{j:S_i<S_j} d_{ij} \wedge \ddot{b}_{j0} + b_{i0} \wedge \sum_{j:S_j<S_i} \ddot{d}_{ji} \right) - \sum_{j=1}^{n} d_{ij} \wedge (m_j \ddot{r}_0 - F_j) = M_i + \sum_{a=1}^{n} S_{ia} Y_a \quad 1 \le i \le n \tag{5.5}$$

Dans ces équations, le terme $\dot{L}_i + \sum_{k=1}^{n} m_k d_{ik} \wedge \ddot{d}_{ik}$ représente le moment angulaire du corps augmenté i par rapport au point de liaison vers le solide s_0, alors que $\dot{L}_i$ désigne

le moment angulaire du corps i par rapport à son barycentre. Les termes où $k \neq i$ sont dus aux $n-1$ masses qui sont attachées au corps augmenté. Si l'on introduit la matrice d'inertie K_i du corps augmenté s_i définie par:

$$K_i = J_i + \sum_{k=1}^{n} m_k \left(d_{ik}^T d_{ik} I_d - d_{ik} d_{ik}^T \right)$$

et ω_i la vitesse angulaire absolue du corps i, le moment angulaire du corps augmenté s_i peut s'écrire sous la forme:

$$\dot{L}_i + \sum_{k=1}^{n} m_k d_{ik} \wedge \ddot{d}_{ik} = K_i \dot{\omega}_i + \omega_i \wedge K_i \omega_i \tag{5.6}$$

D'autre part, en utilisant les propriétés des vecteurs d_{ij} et b_{ij}, à savoir:

$$\sum_{j=1}^{n} d_{ij} \wedge m \ddot{r}_0 = \sum_{j=1}^{n} (b_{i0} - b_{ij}) m_j \wedge \ddot{r}_0 = b_{i0} \wedge M \ddot{r}_0$$

$$\sum_{j=1}^{n} d_{ij} \wedge F_j = \sum_{j:S_j<S_i} d_{ij} \wedge F_j,$$

l'équation (5.5) prend la forme:

$$K_i \dot{\omega}_i + \omega_i \wedge K_i \omega_i + M \left(\sum_{j:S_i<S_j} d_{ij} \wedge \ddot{b}_{j0} + b_{i0} \wedge \left(-\ddot{r}_0 + \sum_{j:S_j<S_i} \ddot{d}_{ij} \right) \right) + \sum_{j:S_i \leq S_j} d_{ij} \wedge F_j = M_i + \sum_{a=1}^{n} S_i a Y_a$$

Les dérivées secondes sont développées de la façon suivante:

$$\begin{aligned} \ddot{b}_{j0} &= \dot{\omega}_j \wedge b_{j0} + \omega_j \wedge (\omega_j \wedge b_{j0}) \\ \ddot{d}_{ji} &= \dot{\omega}_j \wedge d_{ji} + \omega_j \wedge (\omega_j \wedge d_{ji}) \end{aligned}$$

On peut maintenant écrire les équations du mouvement sous une forme plus propice à l'intégration numérique en laissant à gauche les inconnues (les accélérations angulaires):

$$K_i \dot{\omega}_i + M \left(\sum_{j:S_i<S_j} d_{ij} \wedge (\dot{\omega}_j \wedge b_{j0}) + b_{i0} \wedge \sum_{j:S_j<S_i} \dot{\omega}_j \wedge d_{ji} \right) = M_i' + M_i + \sum_{a=1}^{n} S_{ia} Y_a$$

où

$$\begin{aligned} M_i' = & -\omega_i \wedge K_i \omega_i - M \Big(\sum_{j:S_i<S_j} d_{ij} \wedge (\omega_j \wedge (\omega_j \wedge b_{j0})) \\ & + b_{i0} \wedge \left(-\ddot{r}_0 + \sum_{j:S_j<S_i} \omega_j \wedge (\omega_j \wedge d_{ji}) \right) \Big) - \sum_{j:S_i \leq S_j} d_{ij} \wedge F_j \end{aligned}$$

Enfin, en remplaçant les produits vectoriels par leur forme matricielle :

$$d_{ij} \wedge (\dot{\omega}_j \wedge b_{j0}) = (b_{j0}^T d_{ij} Id - b_{j0} d_{ij}^T)\dot{\omega}_j$$
$$b_{i0} \wedge (\dot{\omega}_j \wedge d_{ji}) = (b_{ji}^T . b_{i0} Id - d_{ji} b_{i0}^T)\dot{\omega}_j$$

et en utilisant les n^2 matrices données par :

$$K_{ij} = \begin{cases} K_i & \text{si} \quad i = j \\ M(b_{j0}^T . d_{ij} Id - b_{j0} d_{ij}^T) & \text{si} \quad S_i < S_j \\ M(d_{ji}^T . b_{i0} Id - d_{ji} b_{i0}^T) & \text{si} \quad S_j < S_i \\ 0 & \text{sinon} \end{cases}$$

l'équation finale du mouvement pour un corps s_i s'écrit :

$$\sum_{j=1}^{n} K_{ij}\dot{\omega}_j = M_i' + M_i + \sum_{a=1}^{n} S_{ia} Y_a \tag{5.7}$$

qui a bien la forme $A\ddot{\theta} = B$. Elle peut être utilisée soit pour des applications numériques, soit pour construire les équations symboliques du mouvement. C'est l'objet du paragraphe suivant.

5.3 Deux utilisations des équations du mouvement

5.3.1 Animation par dynamique directe

Employée telle quelle, l'équation précédente peut être utilisée pour générer des simulations dynamiques. Toutes les expressions sont évaluées numériquement pour donner une équation de la forme :

$$A\dot{\omega} = B$$

L'inversion de ce système suivi d'une première intégration permet de retrouver les vitesses angulaires ω. Habituellement, les trois degrés de liberté disponibles sur chaque articulation sont représentés par les angles d'Euler ϕ, θ et ψ. Les trois angles qui caractérisent une liaison a permettent de passer du repère de $s_{i+(a)}$ au repère de $s_{i-(a)}$. Chacun de ces angles définit une rotation par rapport à trois axes mobiles liés au repère local de l'articulation. Par exemple, si ces trois axes sont les axes z, x et z à nouveau, la relation liant ω aux vitesses angulaires des trois rotations élémentaires est donnée par :

$$\omega = \dot{\phi} z + \dot{\theta} x' + \dot{\psi} z'' \tag{5.8}$$

où x' contient les coordonnées de x transformé par la rotation d'angle ϕ et z'' contient les coordonnées de z transformé par les deux premières rotations.

Ici, comme dans le cas des interpolations d'orientation, l'emploi des angles d'Euler, comme système de paramétrage, pose un problème. Ils sont en effet dépendants de l'orientation des axes par rapport auxquels ils sont définis. Dans le cas où $\theta = n\pi$, les axes des

rotations d'angle ϕ et ψ coïncident[3]. Choisir un autre système d'axe que $z-x-z$ ne permet pas d'éliminer le problème. Les singularités se retrouvent, mais sur un autre axe et éventuellement avec une autre valeur particulière.

Si l'on reprend l'équation (5.8) et que l'on décompose les axes dans le deuxième repère (celui de $s_{i-(a)}$), on obtient les trois équations :

$$\begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} = \begin{pmatrix} \sin\theta\sin\psi & \cos\psi & 0 \\ \sin\theta\cos\psi & -\sin\psi & 0 \\ \cos\theta & 0 & 1 \end{pmatrix} \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix}$$

En inversant ce système on obtient les équations différentielles exprimant les dérivées des paramètres en fonction des vitesses angulaires :

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} \frac{\sin\psi}{\sin\theta} & \frac{\cos\psi}{\sin\theta} & 0 \\ \cos\psi & -\sin\psi & 0 \\ -\sin\psi\frac{\cos\theta}{\sin\theta} & -\cos\psi\frac{\cos\theta}{\sin\theta} & 1 \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix}$$

Ces équations montrent de façon évidente que des problèmes se posent quand l'angle θ est un multiple de π. En pratique, ces singularités sont éludées en choisissant les trois axes de rotations adaptés au type de mécanisme utilisé.

De notre côté, nous avons gardé la représentation des orientations sous forme de quaternion. Pour lier un quaternion q à la vitesse angulaire, il existe la relation $\omega = 2q^{-1}\dot{q}$ [RA90]. En notant q sous la forme d'une matrice 4×4, on obtient les équations explicites :

$$\begin{pmatrix} 0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} = \begin{pmatrix} q_w & q_x & q_y & q_z \\ -q_x & q_w & q_z & -q_y \\ -q_y & -q_z & q_w & q_x \\ -q_z & q_y & -q_1 & q_w \end{pmatrix} \begin{pmatrix} \dot{q}_w \\ \dot{q}_x \\ \dot{q}_y \\ \dot{q}_z \end{pmatrix}$$

et en réarrangeant les différents termes, on trouve les équations différentielles donnant $\dot{q}$ en fonction de ω et q :

$$\begin{pmatrix} \dot{q}_w \\ \dot{q}_x \\ \dot{q}_y \\ \dot{q}_z \end{pmatrix} = \begin{pmatrix} 0 & -\omega_1 & -\omega_2 & -\omega_3 \\ \omega_1 & 0 & \omega_3 & -\omega_2 \\ \omega_2 & -\omega_3 & 0 & \omega_1 \\ \omega_3 & \omega_2 & -\omega_1 & 0 \end{pmatrix} \begin{pmatrix} q_w \\ q_x \\ q_y \\ q_z \end{pmatrix} \tag{5.9}$$

Entre deux intervalles de temps, l'intégration de ces quatre équations permet de trouver les nouvelles orientations à partir de la vitesse angulaire et de l'ancienne orientation. Ces quatre équations sont liées par le fait que les quaternions sont normés. Cette contrainte permet – en renormalisant q à la fin de chaque intégration – de corriger les erreurs d'arrondis.

[3] Dans la réalité, ce phénomène de blocage (en anglais, *gimbal lock*) peut survenir sur des suspensions à la Cardan tel que le gyroscope.

Pour résumer, le calcul d'une étape de simulation se fait de la manière suivante :

calculer les matrices A et B
calculer $\dot{\omega} = A^{-1}B$
intégrer l'équation ci-dessus pour trouver ω
intégrer l'équation (5.9) pour trouver q
normer q

Avant de passer à la construction symbolique de ces équations, arrêtons nous un moment sur la méthode d'intégration qui a été utilisée. Dans les sections suivantes, nous serons fréquemment confrontés à l'intégration d'équations différentielles. Ce domaine de l'analyse numérique consiste à déterminer les valeurs d'une fonction $x(t)$ connaissant la dérivée $\dot{x}(t) = f(x,t)$ de cette fonction et la valeur initiale $x_0 = x(t_0)$. Contrairement à ce que nous verrons par la suite, il s'agit ici uniquement d'un problème différentiel avec valeurs initiales.

L'idée fondamentale de toutes les méthodes d'intégration numérique est la discrétisation. Cela permet de calculer les variations de la fonction à partir de petites variations Δt du temps.

La méthode la plus simple, mais aussi la moins efficace, est la méthode d'Euler (ou de la tangente). Elle est basée sur le développement limité à l'ordre un qui approxime la valeur de la fonction à un instant $t+\Delta t$ à partir de la valeur de la fonction et de sa dérivée à l'instant précédent t :

$$x(t+\Delta t) = x(t) + \Delta t f(x,t)$$

Il existe une méthode d'ordre deux appelée méthode du milieu sur laquelle nous passons pour arriver à la méthode d'ordre quatre de Runge-Kutta (sur les algorithmes d'intégration numérique, consulter par exemple [Gea91]). C'est, parmi les méthodes à pas constant, celle qui offre le meilleur compromis entre temps de calcul et efficacité de l'approximation. Le calcul de la fonction à l'instant $t + \Delta t$ est réalisé à partir de quatre évaluations de la fonction dérivée :

$$\begin{aligned} k_1 &= \Delta t f(x,t) \\ k_2 &= \Delta t f(x + \frac{k_1}{2}, t + \frac{\Delta t}{2}) \\ k_3 &= \Delta t f(x + \frac{k_2}{2}, t + \frac{\Delta t}{2}) \\ k_4 &= \Delta t f(x + k_3, t + \Delta t) \end{aligned}$$

La valeur au pas d'intégration suivant est alors donnée par :

$$x(t+\Delta t) = x(t) + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}$$

C'est cette méthode que nous avons utilisée pour les deux étapes d'intégration décrites plus haut. Nous nous en servirons aussi dans le chapitre qui suit. L'inconvénient est qu'elle

requiert quatre évaluations (ici, très coûteuses) de la fonction f à chaque intervalle de temps. Elle est néanmoins reconnue comme offrant le meilleur rapport entre efficacité et coût; en particulier, elle est bien plus stable qu'une méthode d'Euler appliquée à des intervalles de temps divisés par quatre.

5.3.2 Construction des équations symboliques du mouvement

La construction des équations symboliques du mouvement se fait en appliquant directement la formule (5.7). Sans restriction particulière, les expressions résultantes sont gigantesques, même pour de très simples systèmes. Bien sûr, en comparaison, les expressions fournies par le module symbolique de Arnaldi, Dumont et Hégron sont plus simples grâce à l'introduction de paramètres auxiliaires, mais ne permettent pas de faire du contrôle optimal. Dans le cadre des exemples présentés dans cette thèse, nous nous sommes limités à des mouvements plans.

Chaque corps s_i est ainsi paramétré par un angle θ_i qui définit la rotation du corps dans le plan xy. Seule la troisième composante de l'équation (5.7) est prise en compte. Concernant la matrice d'inertie J_i, seul le moment d'inertie par rapport à l'axe z est utilisé. C'est le dernier terme sur la diagonale de J_i.

Pour construire les équations, nous avons développé un module réduit de calcul symbolique. Les expressions sont représentées sous forme d'arbre binaire. Les nœuds peuvent contenir les opérateurs `+`, `-`, `*`, `/` ou les fonctions circulaires `cos`, `sin`. Les feuilles contiennent soit les variables, soit des constantes. Les variables sont, outre les angles θ_i et leurs dérivées $\dot{\theta}_i$, les éléments u_i de la matrice Y. Rappelons que ces variables, qui représentent les couples appliqués à chaque articulation, ne sont pas des données du problème mais seront déterminées par des techniques de contrôle optimal. Les constantes sont directement représentées sous forme numérique, ce qui nécessite de reconstruire les équations quand une caractéristique du solide est modifiée.

Sur cette couche élémentaire, un module de calcul symbolique matriciel a été ajouté. Il permet d'effectuer des sommes et des produits entre matrices symboliques, mais aussi des produits vectoriels. Tous les éléments évoqués au paragraphe 5.2 doivent être construits sous forme symbolique. Par exemple, la matrice d'adjacence S est maintenant une matrice symbolique dont les éléments sont des expressions symboliques contenant les constantes 1, -1 ou 0. Les vecteurs sont stockés dans des matrices à une seule ligne, ou une seule colonne suivant les cas. Cela permet de limiter les types d'opérations entre matrices.

Enfin des fonctions permettant la dérivation partielle ont été ajoutées. Elles seront utilisées dans le prochain chapitre.

Remarque

Pour stocker les expressions symboliques, il aurait été préférable d'utiliser une structure de graphe acyclique orienté . C'est ce que fait Dumont dans [Dum90]; cela permet de partager les expressions symboliques communes. La méthode des travaux virtuels qu'il utilise nécessite d'effectuer de nombreuses dérivations partielles ou par rapport au temps. Celles-ci dupliquent alors de nombreux termes. Contrairement à sa méthode, nous n'utilisons pas – du moins pour construire les équations du mouvement – d'opérateurs de dérivation. Pour l'instant, nous n'utilisons pas de structure de DAG. Ce serait une extension possible.

5.3.3 Mouvement d'un pendule simple

Pour finir ce chapitre, nous donnons l'équation du mouvement d'un pendule simple (figure 5.4). Elle sera à la base des exemples donnés dans le chapitre qui suit.

Les termes qui interviennent dans la description du pendule simple sont:

- sa masse m,
- sa longueur l,
- le moment d'inertie J par rapport à l'axe de rotation z; Ici, $J = \frac{ml^2}{3}$, et la matrice du corps augmenté est $K = J$,
- la matrice d'adjacence $S = (-1)$,
- le vecteur F contenant la force de gravité appliquée en C; $F^T = (0, mg, 0)$,
- le vecteur d reliant C à l'unique articulation; $d^T = (\frac{l}{2}\sin\theta, \frac{l}{2}\cos\theta, 0)$,
- le couple $Y^T = (0, 0, u)$ appliqué sur l'articulation,
- l'accélération angulaire $\dot{\omega}^T = (0, 0, \ddot{\theta})$.

En appliquant l'équation générale (5.7):

$$\frac{ml^2}{3}\begin{pmatrix} 0 \\ 0 \\ \ddot{\theta} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ u \end{pmatrix} - \begin{pmatrix} -\frac{l}{2}\sin\theta \\ -\frac{l}{2}\cos\theta \\ 0 \end{pmatrix} \wedge \begin{pmatrix} 0 \\ mg \\ 0 \end{pmatrix} \tag{5.10}$$

et ne conservant que la troisième composante de cette équation, on retrouve l'équation connue:

$$\frac{ml^2}{3}\ddot{\theta} = u - mg\frac{l}{2}\sin\theta \tag{5.11}$$

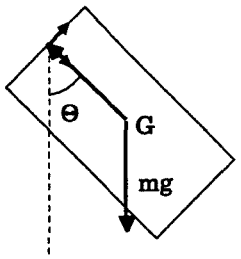


FIG. 5.4 Pendule à un bras

Chapitre 6

Méthodes de minimisation

Ce chapitre a pour objectif de montrer l'emploi qui peut être fait, en animation, des techniques de minimisation. Jusqu'à présent trois approches ont été proposées. Elles utilisent soit soit une méthode d'optimisation pour des équations discrètes du mouvement, soit une méthode combinatoire connue sous le nom de programmation dynamique, soit enfin, le contrôle optimal pour des équations linéaires du mouvement. Nous proposons une nouvelle technique basée sur le contrôle optimal, mais concernant cette fois des équations non linéaires.

Ce chapitre est organisé de la façon suivante :

- Les méthodes d'optimisation discrète et de programmation dynamique sont l'objet des sections 6.1 et 6.2
- Le paragraphe 6.3 expose les principes généraux du contrôle optimal et donne les équations de référence.
- La section 6.4 traite d'une méthode existante limitée aux équations linéaires.
- Dans la section 6.5, nous décrivons l'extension au cas des équations non linéaires du mouvement. Nous y détaillons les aspects théoriques ainsi que la mise en œuvre.

6.1 Méthode d'optimisation

Dans les parties qui suivront, nous aborderons le problème d'un point de vue continu, même si une discrétisation implicite est finalement effectuée lors de l'intégration des différents systèmes différentiels.

Ce paragraphe se base sur les travaux exposés par Witkin et Kass où le problème est vu sous un angle différent [WK88]. Ils partent du principe classique qu'un mouvement peut être discrétisé en un nombre fini d'intervalles de temps : $t_1, \ldots, t_N$. L'état du système était décrit dans le paragraphe précédent par un vecteur $x(t)$. Maintenant, il va être décrit par l'ensemble des paramètres du mouvement à chacune des N étapes : $x_1, x_2, \ldots, x_N$. De

même, les forces intervenant étaient définies par un vecteur $u(t)$ auquel va se substituer l'ensemble des forces entrant en jeu à chaque intervalle de temps, c'est à dire $u_1, u_2, \ldots, u_N$.

Classiquement, la discrétisation du mouvement est effectuée à l'aide des différences finies[1]. Ainsi, en utilisant l'approximation des différences finies, les dérivées premières ou secondes des paramètres généraux du mouvement peuvent s'approximer en fonction des paramètres du mouvement et de l'intervalle de temps :

$$\dot{x}_i = \frac{x_i - x_{i-1}}{\Delta t} \text{ et } \ddot{x}_i = \frac{x_{i-1} - 2x_i + x_{i+1}}{\Delta t^2} \tag{6.1}$$

où $\dot{x}_i$ et $\ddot{x}_i$ désignent respectivement la vitesse et l'accélération du paramètre x_i à l'étape i. Les intervalles de temps Δt sont constants.

Une équation différentielle d'ordre deux est ainsi décomposée en un ensemble de N équations non linéaires. Ces dernières, étant vues comme des contraintes assurant le respect des lois physiques, sont appelées contraintes physiques. Chacune d'elles décrit le passage de l'étape i à l'étape $i+1$. Un avantage de cette formulation est qu'il n'est pas nécessaire de disposer d'équations explicites sous la forme $\ddot{x} = f(\dot{x}, x)$. Une formulation implicite du type $f(\ddot{x}, \dot{x}, x) = 0$ suffit, ce qui permet d'utiliser la méthode de Lagrange pour obtenir les équations du mouvement. En utilisant les différences finies, l'équation précédente est remplacée par les N contraintes physiques $c_i(x_{i-1}, x_i, x_{i+1}) = 0$. En pratique, cette décomposition peut amener à des systèmes gigantesques. Par exemple, l'animation d'un corps composé de cinq degrés de liberté pendant quatre secondes introduit 500 équations de contrainte, si l'on prend 25 échantillons par seconde. En pratique, les équations sont souvent instables et il est préférable de faire un suréchantillonnage temporel.

D'autres contraintes, géométriques cette fois, permettent de définir les valeurs des paramètres aux bornes, mais aussi tout autre type de contrainte. De plus, il est possible d'imposer des contraintes géométriques à tout instant de l'animation.

Il reste à introduire un critère à minimiser. Dans le paragraphe précédent, le phénomène étant traité de manière continue, le critère était sous la forme d'une intégrale. Maintenant il va s'agir de minimiser la somme des énergies dépensées qui peut s'exprimer sous la forme :

$$\sum_{i=1}^{N} f(u_i, x_i) \tag{6.2}$$

La fonction f détermine le type de critère.

Le problème qu'il faut résoudre est donc le suivant : étant donné un ensemble de variables (les x_i et les u_i), minimiser un critère de la forme (6.2) tout en respectant un ensemble de contraintes. Ces dernières incluent indifféremment les contraintes géométriques et physiques.

Contrairement au contrôle optimal où le mouvement était traité séquentiellement, par l'intermédiaire des équations différentielles, ici, le mouvement global est intégré dans la

[1]Cette méthode est aussi utilisé dans [Arn88] pour transformer les équations différentielles du mouvement en un système d'équations non linéaires. Celui-ci est ensuite résolu par l'algorithme de Newton-Raphson afin de donner le mouvement en dynamique directe du solide

formulation du problème. C'est un problème d'optimisation sous contraintes non linéaires. De nombreux algorithmes permettent de résoudre un tel problème. Ils dépendent essentiellement de la connaissance dont on dispose sur la fonction f et sur les contraintes c_i : est on seulement capable d'évaluer f et les c_i à partir des valeurs des variables? Connait-on d'autres informations telles que le gradient, le jacobien?

En pratique, les équations du mouvement sont construites là aussi symboliquement. Un module spécifique permet d'en déduire automatiquement les contraintes physiques associées. Il est donc facile de calculer le gradient et le hessien de f ainsi que le jacobien des contraintes. Ces derniers permettent d'utiliser une méthode numérique performante – la connaissance du hessien assure une convergence quadratique. Les matrices dérivées peuvent atteindre des tailles très importantes (voir l'exemple ci-dessous), mais peu de leurs éléments sont significatifs (matrices creuses). Chaque itération de l'algorithme de minimisation est constituée de deux étapes. La première consiste à assurer une décroissance de la fonction avec les variables libres (voir annexe C). La deuxième assure le respect des contraintes.

Exemple du pendule

Nous reprenons l'exemple du pendule simple dont l'équation du mouvement est donnée par :

$$\frac{ml^2}{3}\ddot{\theta} - u + mg\frac{l}{2}\sin\theta = 0 \tag{6.3}$$

A un instant t_i, et en utilisant les différences finies (voir équation (6.1)), l'équation différentielle (6.3) est remplacée par les N contraintes physiques $c_i(x)$ suivantes :

$$c_i(x) = \frac{ml^2}{3}\frac{\theta_{i-1} - 2\theta_i + \theta_{i+1}}{\Delta t^2} - u_i + \frac{mgl}{2}\sin\theta_i$$

Les variables du problème d'optimisation sont les angles θ_i et les couples u_i. Il faut d'autre part ajouter les contraintes de positions. Par exemple, on peut se contenter de spécifier les valeurs angulaires α et β à t_0 et t_N de la manière suivante :

$$\begin{aligned} c_\alpha &= \theta_0 - \alpha \\ c_\beta &= \theta_N - \beta \end{aligned}$$

Il est possible d'ajouter autant de contraintes qu'on le souhaite. Celles-ci peuvent d'ailleurs être définies à n'importe quel instant compris entre t_0 et t_N. Le jacobien des contraintes c_i $(0 \leq i \leq N)$ c_α et c_β s'écrit alors :

$$\frac{\partial c_i}{\partial \theta_j} = \begin{cases} \frac{1}{\Delta t^2} & \text{si} \quad i = j+1 \text{ ou } i = j-1 \\ \frac{-2ml^2}{3\Delta t^2} + \frac{mgl}{2}\cos\theta_i & \text{si} \quad i = j \\ 0 & \text{sinon} \end{cases}$$

D'autre part, si on définit la fonction critère comme la somme des couples utilisés au cours du mouvement :

$$f = \sum_{i=0}^{N} u_i^2$$

le gradient et le hessien sont:

$$g_i = 2u_i \text{ et } H_{i,j} = \begin{cases} 2 & \text{si } i = j \\ 0 & \text{sinon} \end{cases} .$$

Il reste alors à appliquer l'algorithme permettant de minimiser f en respectant les contraintes c.

6.2 Programmation dynamique

Dans [dPFV90], une autre approche, fondée sur la programmation dynamique, est présentée. L'intérêt principal est la réutilisabilité du mouvement calculé, et ceci grâce à la notion de contrôleur d'état.

Un contrôleur d'état est défini sur un domaine représentant l'ensemble des états que peut prendre le système mécanique. Maintenant, par état, on entend toute combinaison possible des valeurs que peuvent prendre les paramètres du système. Un contrôleur d'état contient tous les chemins optimaux permettant de faire passer le système d'un endroit du domaine à un état final propre au contrôleur. Ainsi, celui-ci peut être utilisé plusieurs fois, à partir d'états initiaux différents. Il est possible de créer des mouvements plus complexes en mettant bout à bout les trajectoires définies par plusieurs contrôleurs agissant en séquence, sous réserve, bien sûr, que les domaines des contrôleurs successifs aient une intersection commune.

Dans la section précédente, la discrétisation était temporelle. A chaque degré de liberté était associée une variable correspondant à un instant donné de la simulation. Maintenant, la discrétisation s'effectue sur les différentes valeurs que peut prendre un paramètre du système. Le domaine contient l'ensemble de ces valeurs. C'est un hypercube de dimension n, où n représente le nombre de paramètres du système. Par exemple, un pendule simple est paramétré par un angle et une vitesse de rotation. Le domaine d'un contrôleur pour ce pendule est donc une grille de dimension deux. Une des coordonnées contient les angles possibles, l'autre les vitesses possibles.

Un contrôleur est une fonction qui à chaque point de la grille associe un vecteur de contrôle permettant d'amener le système de ce point à l'état destination propre au contrôleur en minimisant un critère donné. Contrairement aux solutions présentées auparavant où le seul critère à minimiser était l'énergie dépensée pendant le mouvement, ici, un autre facteur intervient. Il s'agit non seulement de minimiser l'énergie mais aussi de réaliser le mouvement dans un minimum de temps. Aussi, l'intégrale à minimiser a-t-elle la forme suivante:

$$I = t_f + \int_{t_0}^{t_f} KT(t)dt$$

où $T(t)$ représente l'énergie dépensée et K est un coefficient permettant de privilégier l'un des deux facteurs que sont l'énergie ou la durée du mouvement.

Plutôt que de résoudre un ensemble de problèmes de contrôle optimal avec valeurs aux deux bornes entre l'ensemble des états du domaine et l'état destination, les auteurs

utilisent une technique appelée programmation dynamique. La programmation dynamique est particulièment bien adaptée à la résolution de problèmes d'optimisation dans les cas où l'état du système et le vecteur de contrôle ne peuvent prendre qu'un nombre limité de valeurs. Le principe en est le suivant : si un chemin AC est optimal, alors quel que soit le point P sur ce chemin optimal, le chemin PC est optimal. En d'autres termes, si dans un domaine donné, on connait tous les chemins optimaux permettant d'atteindre un état destination D, le chemin optimal allant d'un état S – extérieur au domaine – à l'état D est la concaténation du chemin optimal reliant S à la frontière du domaine et de celui reliant cette frontière à D. L'idée est donc de calculer les chemins optimaux en partant de D jusqu'à retrouver S.

Pour illustrer le fonctionnement de la programmation dynamique, intéressons nous à l'exemple suivant. On dispose d'une grille carrée sur laquelle seuls des déplacements de la gauche vers la droite sont permis. Chaque nombre indiqué sur la figure (6.1.a) indique le coût (contrôle) nécessaire à la transition d'un point à son voisin de droite. Par rapport au cas qui nous intéresse, le problème est grandement simplifié puisque pour passer d'un état à un autre, seules deux valeurs de contrôle sont possibles. Le problème est de trouver le chemin de moindre coût joignant A à B, en se déplaçant uniquement de gauche à droite. Plutôt que d'essayer toutes les routes possibles, l'application du principe de la programmation dynamique revient à considérer le problème à l'envers en cherchant, à partir du point B les chemins optimaux menant à B à partir de tous les points de la grille. La première étape (figure 6.1.b) consiste à marquer les deux voisins de B par les coûts respectifs de chacun des deux chemins. Si on considère maintenant le point à gauche de B, deux chemins sont possibles : l'un passant par le coin marqué 10, l'autre passant par le coin marqué 11. Le coût du chemin inférieur est $11 + 7$, celui du chemin supérieur est $10 + 6$, donc c'est ce dernier trajet qui est optimal. Le point considéré est alors marqué du coût 16 et le chemin optimal est indiqué (par les flèches sur la figure). La procédure est répétée de proche en proche jusqu'à atteindre le point A. Une fois tous les points traités, il suffit alors de partir de A en suivant les flèches. En plus, à partir de chaque coin de la grille, il est possible d'atteindre le point B en un coût minimal; il suffit de suivre les flèches.

Pour revenir au cas général, la génération d'un contrôleur revient à associer à chaque point du domaine les valeurs de contrôle permettant d'atteindre l'état final. La génération d'un contrôleur consiste donc à résoudre un ensemble de problèmes locaux d'optimisation. La valeur de la fonction à minimiser est calculée de proche en proche, en partant de l'état destination.

Pour résoudre le problème local en un point du domaine, on opère de la façon suivante. On anime le système à partir de différentes valeurs discrétisées du vecteur de contrôle jusqu'à atteindre une partie du domaine où la valeur du critère est connue. Le meilleur chemin est celui ou la somme de l'énergie dépensée pour atteindre cette frontière et de celle dépensée pour aller de la frontière à destination est minimale. Evidemment, la plupart des chemins générés n'atteignent pas forcément un coin de la grille (ce sont les seuls endroits où la valeur du critère est connue). Pour y remédier, la valeur du critère est calculée par interpolation n-linéaire entre les coins voisins de la grille.

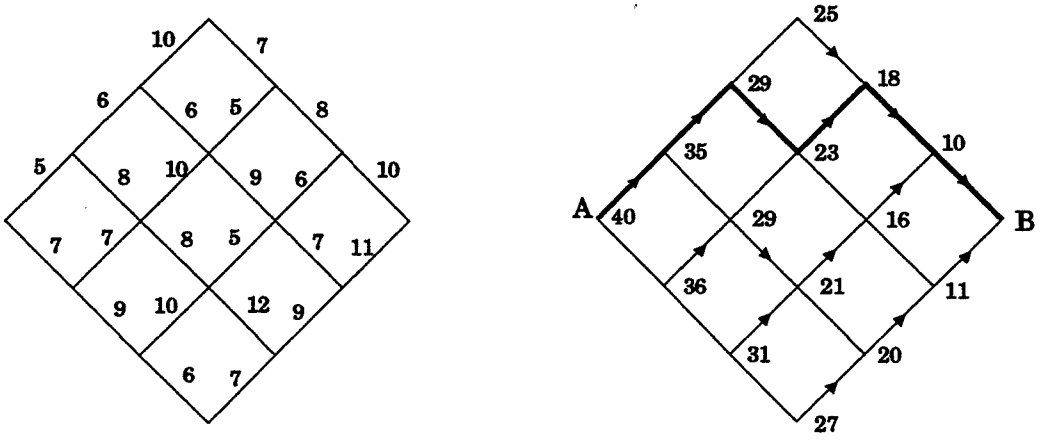


FIG. 6.1 Exemple d'utilisation de la programmation dynamique

6.3 Principes du contrôle optimal

Le contrôle optimal est une technique de calcul des variations employée depuis plusieurs décennies et dont les applications sont fort nombreuses. Citons par exemple le guidage de systèmes aéronautiques où l'économie du carburant utilisé est essentielle, l'aérodynamisme, etc... (cf [BH75]).

Globalement, l'objectif du contrôle optimal est de déterminer le chemin idéal que doit suivre un mobile pour atteindre un objectif donné tout en minimisant (ou maximisant, selon les applications) un critère particulier. La trajectoire est définie par une équation différentielle. L'objet en mouvement influe sur la trajectoire au moyen de forces et de couples. Contrairement à la simulation où les forces sont connues à l'avance, il s'agit ici de les calculer de telle sorte qu'elles permettent d'atteindre l'objectif. Dans le premier cas, le mouvement est complètement déterminé par les positions initiales (en anglais, *initial value problem*). Ici, c'est un problème bien plus complexe puisqu'il faut respecter les positions aux deux bornes (*two points boundary value problem*).

Les données d'un problème de contrôle optimal sont les suivantes :

- L'état du mobile est décrit par un vecteur contenant l'ensemble de ses paramètres, c'est à dire les degrés de liberté. Quand l'équation dirigeant le mouvement est d'ordre deux, on verra plus loin que l'état contient aussi les dérivées de ces paramètres. Dans le cas évoqué précédemment d'un pendule simple, l'état du système est un vecteur bi-dimensionnel contenant l'angle du pendule par rapport à la verticale et sa vitesse de rotation. Par la suite, ce vecteur sera appelé état du système et noté $x(t)$.

- D'autre part, un vecteur de contrôle permet de diriger l'évolution du mobile. Ce vecteur contient en particulier les forces et couples "contrôlables" agissant sur l'objet considéré à chaque intervalle de temps. La pesanteur est une force fixe n'intervenant pas dans le vecteur de contrôle. Dans le cas de notre pendule, il s'agit du couple permettant de le faire pivoter autour de son axe. On notera ce vecteur $u(t)$.

- L'objet en mouvement évolue selon un système d'équations différentielles. Il y a une équation par degré de liberté. Ces équations différentielles décrivent l'évolution du vecteur d'état. Le temps y est traité implicitement.

- Enfin, le dernier élément d'un problème de contrôle optimal est le critère que l'on cherche à minimiser. Il peut dépendre du vecteur d'état, du vecteur de contrôle et du temps. Il s'exprime comme une intégrale de tous ces paramètres au cours du temps.

Les termes employés étant précisés, le problème à résoudre peut s'énoncer sous la forme suivante :

Etant donné un système différentiel de la forme :

$$\dot{x}(t) = f(x(t), u(t), t)$$

évoluant entre un instant initial t_0 et un instant final t_f, et étant données les positions $x(t_0) = x_0$ et $x(t_f) = x_f$, il s'agit de déterminer le vecteur $u(t)$ minimisant au cours du mouvement un scalaire de la forme :

$$J = \Phi(x(t_f)) + \int_{t_0}^{t_f} L(x(t), u(t), t)dt$$

$x(t)$ est le vecteur de dimension n décrivant l'état du système.

$u(t)$ est le vecteur de dimension m (si l'on suppose que chaque degré de liberté est contrôlable, les tailles des vecteurs x et u sont identiques) décrivant le contrôle utilisé lors du mouvement.

$L(x(t), u(t), t)$ est la fonction scalaire appelée critère. La fonction scalaire Φ introduit une contrainte dépendante de l'état du système à l'instant final. Sans intérêt immédiat, elle servira plus tard à prendre en compte les contraintes terminales $x(t_f) = x_{t_f}$.

On vient de le voir dans le chapitre précédent, la construction des équations du mouvement d'un système soumis aux lois de la dynamique conduit à des systèmes d'équations différentielles d'ordre deux. Or ici, les équations considérées sont d'ordre un. En fait, cela n'introduit aucune restriction. Tout système d'ordre deux peut se ramener à un système d'ordre un par un changement de variable approprié.

Pour le système d'ordre deux $\ddot{X} = F(\dot{X}, X, u, t)$, il suffit de prendre comme nouvel état le vecteur $x = (X, \dot{X})$. Cela revient à traiter le système d'ordre un (composé de deux fois plus d'équations) suivant :

$$\dot{x} = F(x, u, t) = \begin{cases} \dot{X} \\ F(X, \dot{X}, t) \end{cases}$$

Les équations de la mécanique étant d'ordre deux, nous utiliserons ultérieurement cette paramétrisation. L'état sera constitué des angles et des vitesses de rotation. D'autre part, comme chaque paramètre du système mécanique possède un couple associé, le vecteur d'état est deux fois plus grand que le vecteur de contrôle ($n = 2m$).

Remarque

Nous n'abordons pas les problèmes de minimisation avec temps final (t_f) non déterminé. Il existe cependant des techniques (dérivées de celles que nous allons décrire) qui permettent de résoudre des problèmes de contrôle optimal où la durée de la simulation intervient dans le critère à minimiser.

Nous allons commencer par donner les équations d'Euler-Lagrange qui définissent les conditions d'optimalité, c'est à dire les conditions qui permettent d'atteindre une intégrale minimale, donc une variation nulle de cette intégrale. Pour cela, nous ne tenons pas compte pour l'instant des conditions finales du type $x(t_f) = x_{t_f}$.

Les équations du mouvement sont intégrées à J grâce au multiplicateur de Lagrange $\lambda(t)$[2]. C'est un vecteur dont la taille égale celle du système (la taille du système dépend du nombre de degrés de liberté de celui-ci mais aussi de l'ordre des équations différentielles). L'intégrale à minimiser est alors :

$$J = \Phi(x(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t) + \lambda^T(t)\left[f(x(t), u(t), t) - \dot{x}(t)\right] dt \qquad (6.4)$$

Pour plus de concision, on utilise la fonction scalaire H, appelée hamiltonien du système :

$$H(x(t), u(t), \lambda(t), t) = L(x(t), u(t), t) + \lambda^T(t)(f(x(t), u(t), t))$$

qui permet de réécrire J de la façon suivante :

$$J = \Phi(x(t_f), t_f) + \int_{t_0}^{t_f} \left(H(x(t), u(t), t) - \lambda^T(t)\dot{x}(t)\right) dt \qquad (6.5)$$

L'intégration par partie du terme $\lambda^T(t)\dot{x}(t)$ de l'équation précédente aboutit à :

$$\begin{aligned} J &= \Phi(x(t_f), t_f) - \lambda^T(t_f)x(t_f) + \lambda^T(t_0)x(t_0) \\ &\quad + \int_{t_0}^{t_f} \left(H(x(t), u(t), \lambda(t), t) + \dot{\lambda}^T(t)x(t)\right) dt \end{aligned}$$

Pour trouver les conditions permettant d'obtenir un minimum sur J, donc d'annuler toute variation de J due à une variation du vecteur de contrôle, commençons par évaluer[3] la variation sur J engendrée par une petite variation du vecteur de contrôle δu :

$$\begin{aligned} \delta J &= \left(\left(\frac{\partial \Phi}{\partial x(t_f)} - \lambda^T(t_f)\right)\delta x(t_f)\right) + (\lambda^T(t_0)\delta x(t_0)) \\ &\quad + \int_{t_0}^{t_f} \left(\left(\frac{\partial H}{\partial x} + \dot{\lambda}^T(t)\right)\delta x(t) + \left(\frac{\partial H}{\partial u}\right)\delta u(t)\right) dt \end{aligned} \qquad (6.6)$$

[2]Remarquons que les multiplicateurs de Lagrange sont aussi employés dans la construction des équations dynamiques du mouvement par le formalisme de Lagrange. Ils permettent d'intégrer des contraintes à l'intérieur des équations du mouvemement.

[3]On utilise des règles de calcul différentiel telles que :

$$\delta H = \frac{\partial H}{\partial u}\delta u + \frac{\partial H}{\partial x}\delta x$$

Le problème étant de trouver une variation de $\delta u(t)$ permettant de minimiser le critère J, on choisit $\lambda(t)$ afin de faire disparaître δx à l'intérieur de l'intégrale :

$$\dot{\lambda}^T = -\frac{\partial H}{\partial x} \tag{6.7}$$

L'annulation du terme ou intervient $\delta x(t_f)$ définit les contraintes finales de l'équation différentielle précédente :

$$\lambda^T(t_f) = \frac{\partial \Phi}{\partial x(t_f)} \tag{6.8}$$

L'équation (6.6) se réduit alors au scalaire suivant :

$$\delta J = \lambda^T(t_0)\delta x(t_0) + \int_{t_0}^{t_f} \frac{\partial H}{\partial u}\,\delta u(t)\,dt \tag{6.9}$$

$\lambda^T(t_0)$ est le gradient de J par rapport aux conditions initiales quand u est constant. Seules les variations de J en fonction des variations de u ont un intérêt. Le premier terme de l'expression (6.9) n'a donc pas à être considéré. Pour trouver un extremum de J, δJ doit être nul pour tout $\delta u(t)$, ce qui fournit la condition d'optimalité sur u :

$$\frac{\partial H}{\partial u} = 0 \tag{6.10}$$

Les équations (6.7) (6.8) et (6.10), connues sous le nom d'équations d'Euler-Lagrange pour le calcul des variations, définissent les conditions d'optimalité d'un système différentiel. La recherche d'un minimum, quand une seule contrainte finale est imposée, nécessite donc la résolution des équations suivantes (H est remplacé par $L + \lambda f$) :

$$\dot{\lambda} = -\left(\frac{\partial f}{\partial x}\right)^T \lambda - \left(\frac{\partial L}{\partial x}\right)^T \text{ avec } \lambda^T(t_f) = \frac{\partial \Phi}{\partial x(t_f)} \tag{6.11}$$

$$\left(\frac{\partial f}{\partial u}\right)^T \lambda + \left(\frac{\partial L}{\partial u}\right)^T = 0 \tag{6.12}$$

6.4 Equations linéaires

Ce paragraphe, inspiré des travaux de Brotman et Netravali [BN88], est consacré à un problème particulier de contrôle optimal. Brotman et Netravali se limitent en effet au cas où le système d'équations différentielles régissant le mouvement est linéaire. D'autre part, le critère à minimiser est choisi quadratique. Ces restrictions permettent d'utiliser des techniques de contrôle particulières dont l'avantage primordial est la rapidité. Les méthodes qui ont été presentées dans les paragraphes 6.1 et 6.2, ainsi que celle présentée ultérieurement dans le paragraphe 6.5, font appel à des algorithmes itératifs qui tendent à approcher (linéairement ou quadratiquement, suivant les cas) la solution. La méthode suivante permet au contraire d'atteindre la solution en une seule étape.

6.4.1 Formalisation du problème

Nous nous situons dans le cas où les équations du mouvement sont définies par un système différentiel linéaire. Le système, qui s'exprime dans le cas général sous la forme $\dot{x} = f(x(t), u(t), t)$, peut maintenant se formuler de manière plus restrictive de la façon suivante :

$$\dot{x}(t) = F(t)x(t) + G(t)u(t) \tag{6.13}$$

$F(t)$ et $G(t)$ sont des matrices de tailles respectives $n \times n$ et $n \times m$ dans lesquelles n'intervient que le temps.

Le critère étant supposé quadratique, on l'écrit maintenant :

$$L(x(t), u(t), t) = \frac{1}{2}\left(x^T(t)Ax(t) + u^T(t)Bu(t)\right) \tag{6.14}$$

L'intégrale à minimiser peut donc s'écrire sous la forme :

$$J = \Phi(x(t_f) + \frac{1}{2}\int_{t_0}^{t_f} \left(x^T(t)Ax(t) + u^T(t)^T Bu(t)\right) dt \tag{6.15}$$

où A et B sont des matrices définies positives de tailles respectives $n \times n$ et $m \times m$ qui permettent d'influer sur le choix du critère à minimiser.

L'hamiltonien correspondant à ce système est :

$$H = \frac{1}{2}x^T(t)Ax(t) + u^T(t)Bu(t) + \lambda^T(t)\left[F(t)x(t) + G(t)u(t)\right]$$

Rappelons que les conditions d'optimalité définies au paragraphe 6.3 sont :

$$\dot{\lambda^T}(t) = -\frac{\partial H}{\partial x}$$

$$\frac{\partial H}{\partial u} = 0$$

Dans le cas linéaire, elle se traduisent par :

$$\frac{\partial H}{\partial x} = \frac{\partial L}{\partial x} + \lambda^T \frac{\partial f}{\partial x} = x^T A + \lambda^T F$$

$$\frac{\partial H}{\partial u} = \frac{\partial L}{\partial u} + \lambda^T \frac{\partial f}{\partial u} = u^T B + \lambda^T G$$

et les équations d'Euler-Lagrange s'écrivent :

$$\begin{aligned} \dot{\lambda} &= -Ax - F^T\lambda \\ u &= -B^{-1}G^T\lambda \end{aligned} \tag{6.16}$$

En remplaçant la valeur de u donnée par (6.16) dans l'équation (6.13), cela revient à résoudre le système différentiel linéaire en x et λ :

$$\begin{cases} \dot{\lambda} = -Ax - F^T\lambda \\ \dot{x} = Fx - GB^{-1}G^T\lambda \end{cases}$$

ou, sous forme matricielle :

$$\begin{pmatrix} \dot{x} \\ \dot{\lambda} \end{pmatrix} = \begin{pmatrix} F & -GB^{-1}G^T \\ -A & -F^T \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} \tag{6.17}$$

avec les conditions aux bornes suivantes :

$$x(t_0) \text{ donné et } \lambda^T(t_f) = \frac{\partial \Phi}{\partial x(t_f)}$$

Pour l'instant, une seule contrainte terminale est spécifiée. En fait, pour intégrer au problème les positions finales $x_{i_{t_f}}$ des n paramètres, n nouveaux multiplicateurs de Lagrange notés ν_i sont utilisés. Les contraintes finales sont sous la forme $\phi_i(t_f) = x_i(t_f) - x_{i_{t_f}}$. $\phi(t_f)$ est le vecteur contenant l'ensemble des contraintes de positions. On choisit donc $\Phi(x_t) = \nu^T \phi(t_f)$ et l'intégrale à minimiser devient :

$$J = \nu\phi(t_f) + \frac{1}{2}\int_{t_0}^{t_f} x^T Ax + u^T Bu \, dt$$

Les conditions terminales sur λ sont alors $\lambda(t_f) = \nu$.

Sans détailler, le problème avec valeurs aux deux bornes revient à résoudre, en partant des conditions finales, les matrices de Riccati suivantes :

$$\begin{array}{ll} \dot{S} + SF + F^T S - SGB^{-1}G^T S = 0 & S(t_f) = 0 \\ \dot{R} + F^T R - SGB^{-1}G^T R = 0 & R^T(t_f) = \frac{\partial \Phi}{\partial x(t_f)} = \mathrm{Id} \\ \dot{Q} - R^T GB^{-1}G^T R = 0 & Q(t_f) = 0 \end{array}$$

Les 0 représentent des matrices composées uniquement de 0, Id désigne l'identité de $\mathcal{R}_n$. Les valeurs de ces trois matrices à $t = t_0$ permettront de trouver les valeurs initiales de λ permettant d'intégrer le système (6.17). D'autre part, les multiplicateurs de Lagrange ν sont donnés par :

$$\nu = Q(t_0)^{-1}\left(\phi - R^T(t_0)x(t_0)\right) \tag{6.18}$$

6.4.2 Algorithme

L'objectif étant de déterminer un vecteur $\lambda(t_0)$ fournissant une condition initiale permettant l'intégration de $\lambda(t)$, il est nécessaire de trouver les valeurs initiales $S(t_0)$, $R(t_0)$ et $Q(t_0)$. Pour cela, Brotman et Netravali commencent par intégrer en parallèle et en arrière les trois équations suivantes, c'est à dire en partant des conditions finales données à droite :

$$\begin{array}{ll} \dot{S} = -(SF + F^T S - SGB^{-1}G^T S) & S(t_f) = 0 \\ \dot{R} = -F^T R + SGB^{-1}G^T R & R^T(t_f) = \mathrm{Id} \\ \dot{Q} = R^T GB^{-1}G^T R = 0 & Q(t_f) = 0 \end{array} \tag{6.19}$$

Seules les valeurs à $t = t_0$ ont un intérêt. A partir des valeurs de $Q(t_0)$ et $R(t_0)$ et de l'équation (6.18) on en déduit la valeur de ν puis la valeur initiale de λ donnée par :

$$\lambda(t_0) = \left(S(t_0) + R(t_0)Q^{-1}(t_0)R(t_0)\right)x(t_0) + R(t_0)Q^{-1}(t_0)\phi \tag{6.20}$$

$x(t_0)$ étant une donnée du problème, $\lambda(t_0)$ étant connue, il reste à intégrer en avant le système différentiel :

$$\begin{pmatrix} \dot{x} \\ \dot{\lambda} \end{pmatrix} = \begin{pmatrix} F & -GB^{-1}G^T \\ -A & -F^T \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix}$$

Le résultat est un ensemble de valeurs $x(t)$ qui représentent l'interpolation de l'état du système entre $x(t_0)$ et $x(t_f)$. L'intérêt fondamental est que ces valeurs sont trouvées en une seule étape, alors que dans le cas d'équations non linéaires, l'algorithme que nous verrons plus loin fait appel à une méthode itérative. Notons qu'il est possible d'estimer les forces utilisées en substituant la valeur de $\lambda(t)$ dans l'équation (6.16) :

$$u(t) = -\left(B^{-1}G^T(S - RQ^{-1}R^T)\right)x(t) - B^{-1}G^T RQ^{-1}\phi$$

6.4.3 Application à l'animation

Seuls des systèmes d'équations linéaires sont pris en compte par la méthode qui vient d'être exposée. Aussi ne peut-on l'appliquer à un grand nombre de modèles dynamiques. En particulier, nous n'appliquerons pas cette approche au cas du pendule dont les équations sont non-linéaires. Nous reprenons donc l'exemple présenté dans [BN88].

Il s'agit d'un point de masse m se déplaçant le long de l'axe des x et soumis à une force de frottement proportionnelle à la vitesse du point. L'état du système est représenté par un vecteur $s(t)$ dont les deux composantes sont la position $x(t)$ sur cet axe et la vitesse $v(t) = \dot{x}(t)$. Le vecteur de contrôle est réduit à un scalaire $u(t)$ que l'on peut considérer comme un accélérateur ou un frein, suivant le signe de $u(t)$.

L'équation du mouvement s'écrit :

$$m\ddot{x} = -\mu\dot{x} - u(t)$$

ou, pour avoir une forme analogue à (6.13) :

$$\dot{s}(t) = \begin{pmatrix} 0 & 1 \\ 0 & -\frac{\mu}{m} \end{pmatrix} s(t) + \begin{pmatrix} 0 \\ -\frac{1}{m} \end{pmatrix} u(t)$$

Les contraintes aux bornes sont $x(t_0) = x_0$, $x(t_f) = t_f$ et $v(t_f) = v_f$. Si l'on veut minimiser le contrôle dépensé et, d'une manière moindre, la vitesse au cours du mouvement, on peut par exemple choisir le critère suivant :

$$L(s(t), u(t)) = v^2(t) + 5u^2(t)$$

Les matrices A et B de la formule (6.15) s'écrivent dans ce cas :

$$A = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, B = (5)$$

Il reste alors à appliquer l'algorithme exposé ci-dessus.

Cette méthode permet de calculer des trajectoires entre deux instants t_0 et t_f où sont spécifiées des contraintes. En général, l'animateur définit un ensemble de contraintes à des instants $t_0, t_1, \ldots, t_n$. Dans ce cas, il y a $n-1$ problèmes à résoudre entre les intervalles t_i et $t_i + 1$ avec $0 \leq i < n$.

Brotman et Netravali montrent que les forces de contrôle utilisées entre deux intervalles successifs sont discontinues, ce qui correspond rarement à la réalité. Pour remédier à cet inconvénient, ils suggèrent d'ajouter le vecteur de contrôle dans le vecteur contenant l'état du système. La variation $w(t)$ du vecteur $u(t)$ tient maintenant le rôle de contrôleur. Ceci permet d'ajouter aux contraintes précédentes sur $s(t)$ des contraintes sur $u(t)$ afin d'assurer la continuité des forces utilisées entre les différentes interpolations.

L'énoncé du problème défini par les équations (6.13) et (6.15) est modifié de la façon suivante :

- Le système d'équations différentielles reste linéaire mais s'écrit maintenant :

$$\dot{\bar{s}}(t) = \begin{pmatrix} F(t) & G(t) \\ 0 & 0 \end{pmatrix} \bar{s}(t) + \begin{pmatrix} 0 \\ w(t) \end{pmatrix}$$

  où $\bar{s}(t) = \begin{pmatrix} s(t) \\ u(t) \end{pmatrix}$ désigne le nouvel état.

- Le critère à minimiser s'écrit maintenant :

$$\bar{L}(x, u, t) = \bar{s}^T(t)\bar{A}\bar{s}(t) + w^T(t)\bar{B}w(t)$$

où $\bar{A}$ et $\bar{B}$ sont les nouvelles matrices déduites des matrices A et B de l'ancien critère (6.15).

6.5 Equations non linéaires

Brotman et Netravali ont appliqué leur méthode à des exemples simples d'animation. Le fait d'utiliser des équations linéaires limite évidemment les types de mouvements possibles. Dans cette partie, nous étendons leur solution au cas des équations non linéaires. Nous commencerons par détailler les résultats du contrôle optimal dans le cas de systèmes régis par des équations non linéaires. Dans cette partie, deux livres [BH75][Ber76] nous ont été particulièment utiles. Nous exposerons ensuite l'algorithme qui a été implanté avant de discuter des résultats obtenus.

6.5.1 Théorie

Dans le paragraphe (6.3) donnant les équations d'Euler-Lagrange, la seule contrainte terminale prise en compte était la fonction scalaire $\Phi(x(t_f))$. Maintenant, comme dans le paragraphe précédent, nous avons une contrainte terminale $\phi_i(x_i(t_f))$ pour chacun des éléments du vecteur d'état. Typiquement, ϕ_i permet de spécifier la valeur du paramètre x_i à l'instant final. C'est une contrainte du type $\phi_i(x_i(t_f)) = x_i(t_f) - x_{i_{t_f}}$.

Si seule la position $x_i(t_f)$ doit être vérifiée, on est amené à résoudre un problème de contrôle réduit où le critère réduit est donné par :

$$K_i = \phi_i(t_f)$$

c'est à dire $L(x,u,t) = 0$ et $\Phi = \phi_i(t_f)$.

Dans ce nouveau problème de minimisation, on utilise maintenant le multiplicateur de Lagrange noté μ_i. C'est un vecteur de la taille du système. Il joue le même rôle que le multiplicateur λ introduit auparavant pour la définition de l'Hamiltonien :

$$H_i = L + \mu_i^T f = \mu_i^T f$$

Les équations d'Euler-Lagrange appliquées à ce sous-problème nous donnent à partir de (6.9) :

$$\delta K_i = \delta\phi_i(t_f) = \int_{t_0}^{t_f} \mu_i^T \frac{\partial f}{\partial u}\delta u(t)\,dt \tag{6.21}$$

avec

$$\dot{\mu}_i = -\left(\frac{\partial f}{\partial x}\right)^T \mu_i$$

et, comme $\phi(t_f)$ ne dépend que de $x_i(t_f)$, les contraintes finales sont[4] :

$$\mu_{i,j}(t_f) = \frac{\partial \phi_i}{\partial x_j} = \begin{cases} 0 & \text{si} \quad i \neq j \\ 1 & \text{sinon} \end{cases}$$

On revient maintenant au critère J du départ. Les n critères K_i y sont ajoutés en utilisant une constante vectorielle ν. Ce nouveau critère sera noté $\bar{J}$. Les multiplicateurs μ_i sont rangés dans la matrice diagonale μ. Pour déterminer une nouvelle variation $\delta u(t)$ assurant une décroissance de J tout en respectant les contraintes terminales, on écrit la variation conjuguée des intégrales K_i et de l'intégrale J :

$$\begin{aligned} \delta\bar{J} &= \delta J + \nu^T \delta\phi(t_f) \\ &= \int_{t_0}^{t_f} \left(\frac{\partial L}{\partial u} + \lambda^T \frac{\partial f}{\partial u}\right)\delta u(t)\,dt + \nu^T \int_{t_0}^{t_f} \left(\mu^T \frac{\partial f}{\partial u}\delta u(t)\right)\,dt \\ &= \left(\frac{\partial L}{\partial u} + (\lambda + \mu\nu)^T \frac{\partial f}{\partial u}\right)\delta u\,dt \end{aligned} \tag{6.22}$$

On choisit maintenant δu de manière à assurer une variation négative de J :

$$\delta u = -\left(\left(\frac{\partial f}{\partial u}\right)^T (\lambda + \nu\mu) + \left(\frac{\partial L}{\partial u}\right)^T\right) \tag{6.23}$$

[4] $\mu_{i,j}$ désigne la $j^{\text{ème}}$ composante de μ_i

Ce vecteur, qui est le gradient de J augmenté des contraintes terminales, correspond à la direction de plus grande descente. En remplaçant cette expression dans (6.22), on vérifie la décroissance du critère augmenté des contraintes finales :

$$\delta\bar{J} = -\int_{t_0}^{t_f} \left\| \left(\frac{\partial f}{\partial u}\right)^T (\lambda + \nu\mu) + \left(\frac{\partial L}{\partial u}\right)^T \right\| dt < 0$$

En substituant (6.23) dans l'équation (6.21) considérée pour toutes les contraintes, on trouve :

$$\delta\phi = A\nu - B$$

où A est la matrice carrée d'ordre n :

$$A = \int_{t_0}^{t_f} \mu^T \frac{\partial f}{\partial u}\left(\frac{\partial f}{\partial u}\right)^T \mu\, dt,$$

et B le vecteur de dimension n :

$$B = \int_{t_0}^{t_f} \mu^T \frac{\partial f}{\partial u}\left(\left(\frac{\partial f}{\partial u}\right)^T \lambda + \left(\frac{\partial L}{\partial u}\right)^T\right) dt$$

La constante vectorielle ν est alors donnée par :

$$\nu = A^{-1}(\delta\phi + B) \tag{6.24}$$

Il reste à définir les conditions permettant d'arrêter l'algorithme. Pour détecter que la solution optimale a été trouvée, il faut déterminer le moment où $\delta\bar{J}$ s'annule. En fait les contraintes étant vérifiées à $t = t_f$, il suffit de regarder si δJ s'annule. Pour cela, on substitue les expressions de $\delta u(t)$ et ν dans l'équation (6.9). La variation du critère s'écrit maintenant :

$$\begin{aligned}
\delta J &= -\int_{t_0}^{t_f} \left(\frac{\partial L}{\partial u} + \lambda^T \frac{\partial f}{\partial u}\right)\left[\left(\frac{\partial f}{\partial u}\right)^T (\lambda - A^{-1}(\delta\phi + B)\mu) + \left(\frac{\partial L}{\partial u}\right)^T\right] dt \\
&= -\int_{t_0}^{t_f} \left(\frac{\partial L}{\partial u} + \lambda^T \frac{\partial f}{\partial u}\right)\left(\left(\frac{\partial f}{\partial u}\right)^T \lambda + \left(\frac{\partial L}{\partial u}\right)^T\right) \\
&\quad + \left(\frac{\partial L}{\partial u} + \lambda^T \frac{\partial f}{\partial u}\right)\left(\frac{\partial f}{\partial u}\right)^T A^{-1} B\mu + \left(\frac{\partial L}{\partial u} + \lambda^T \frac{\partial f}{\partial u}\right)\left(\left(\frac{\partial f}{\partial u}\right)^T A^{-1}\delta\phi\mu\right) dt
\end{aligned}$$

Finalement, en définissant :

$$C = \int_{t_0}^{t_f} \left(\frac{\partial L}{\partial u} + \lambda^T \frac{\partial f}{\partial u}\right)\left(\left(\frac{\partial f}{\partial u}\right)^T \lambda + \left(\frac{\partial L}{\partial u}\right)^T\right) dt$$

on peut exprimer la variation de l'intégrale J en fonction des trois intégrales A, B et C sous la forme :

$$\delta J = -C + B^T A^{-1} B + B^T A^{-1}\delta\phi$$

A partir de là, il est clair que l'on va se rapprocher de l'optimum quand d'une part les contraintes seront satisfaites ($\delta\phi \rightarrow 0$) et d'autre part :

$$-C + B^T A^{-1} B \rightarrow 0$$

Tous ces développements sont récapitulés et ordonnés de manière à être utilisés sous forme d'algorithme. Les multiplicateurs de Lagrange notés jusqu'à présent λ (respectivement μ) sont représentés par le vecteur p (respectivement par la matrice R). R est une matrice diagonale, on ne distinguera donc pas R^T de R.

6.5.2 L'algorithme

Nous donnons ici un aperçu schématique de l'algorithme qui a été implanté. Les différentes variables temporelles ($x(t)$, $u(t)$) sont utilisées sous une forme discrète. Les formes numériques des divers matrices et gradients sont obtenues par évaluation des expressions symboliques associées. Des détails plus complets sur l'implantation de cet algorithme sont fournis au paragraphe 6.5.4.

1) Estimer un ensemble de valeurs pour $u(t)$.

2) Intégrer les équations du mouvement $\dot{x}(t) = f(x(t), u(t), t)$ à partir des conditions initiales $x(t_0)$ et des valeurs de $u(t)$ définies ci-dessus. Stocker $x(t)$ et $\phi(x(t_f))$.

3) Intégrer en arrière la fonction vectorielle $\lambda(t)$ et la matrice $\mu(t)$:

$$\dot{p}(t) = -\frac{\partial f}{\partial x} p(t) - \left(\frac{\partial L}{\partial x}\right)^T \quad p(t_f) = 0$$

$$\dot{R}(t) = -\left(\frac{\partial f}{\partial x}\right)^T \quad R(t_f) = Id$$

4) Calculer les intégrales suivantes :

$$A = \int_{t_0}^{t_f} R \frac{\partial f}{\partial u} \left(\frac{\partial f}{\partial u}\right)^T R \, dt$$

$$B = \int_{t_0}^{t_f} R^T \frac{\partial f}{\partial u} \left(\left(\frac{\partial f}{\partial u}\right)^T p + \left(\frac{\partial L}{\partial u}\right)^T \right) dt$$

$$C = \int_{t_0}^{t_f} \left(\frac{\partial L}{\partial u} + p^T \frac{\partial f}{\partial u}\right) \left(\left(\frac{\partial f}{\partial u}\right)^T p + \left(\frac{\partial L}{\partial u}\right)^T \right) dt$$

5) Choisir des valeurs de $\delta\phi$ de façon à amener progressivement $\phi(x(t_f))$ proche de 0 :

$$\delta\phi = -\epsilon\phi(x(t_f))$$

6) Calculer ν :

$$\nu = -A^{-1}(\delta\phi + B)$$

7) Mettre à jour le vecteur de contrôle

$$u(t) = u(t) + \delta u(t)$$

où

$$\delta u(t) = -\left(\frac{\partial L}{\partial u} + (p(t) + R(t)\nu)^T \frac{\partial f}{\partial u}\right)^T$$

Les étapes (2) à (7) sont répétées jusqu'à ce que la condition d'optimalité soit vérifiée, c'est à dire quand $\phi(x(t_f))$ et $C - B^T A^{-1} B$ sont suffisamment proches de zéro.

6.5.3 Application à l'animation

Cet algorithme utilise des résultats théoriques du calcul des variations. En particulier, les systèmes d'équations sont continus. Pour pouvoir appliquer cet algorithme, il faut néammoins se ramener en fin de compte à une discrétisation. Le temps est donc décomposé en un ensemble de $N+1$ échantillons temporels $t_i = \frac{1-i}{N} t_0 + \frac{i}{N}(t_f)$. Cette discrétisation explique a posteriori que nous utilisions la méthode de Runge-Kutta (dont le pas d'intégration est fixe). En effet, une méthode à pas adaptatif nécessite de pouvoir modifier l'échantillonnage temporel au cours de l'intégration, ce qui n'est manifestement pas possible ici, d'autant plus que nous avons trois intégrations différentes à mener (x, p et R). En pratique, nous avons donc choisi un pas de temps faible pour éviter les problèmes d'instabilité numérique. Dans les exemples donnés plus loin, une seconde d'animation est échantillonnée par 100 intervalles de temps.

Avant de revoir en détail la mise en pratique des sept étapes de l'algorithme, il faut parler de l'utilisation des équations symboliques du mouvement établies dans le chapitre précédent. Nous en étions restés à la construction symbolique des matrices A et B. Pour obtenir une équation de la forme $\dot{x}(t) = f(x(t), u(t), t)$, il faut inverser A. Symboliquement, les calculs sont compliqués. Aussi, l'inversion symbolique n'a été réalisée que pour des systèmes composés de deux équations (deux degrés de liberté). Nous continuerons néanmoins à discuter de cet algorithme dans le cas général où le système mécanique est constitué de n paramètres.

Les équations formelles contiennent des expressions dépendant des variables θ_i, $\dot{\theta}_i$ et u_i. Les θ_i et $\dot{\theta}_i$ constituent l'état du système, les u_i constituent le vecteur de contrôle. En utilisant la paramétrisation proposée au début du chapitre, on a donc $x = (\theta_1, \dot{\theta}_1, \ldots, \theta_n, \dot{\theta}_n)$ et $u = (u_1, \ldots, u_n)$. A chaque paramètre θ_i est associée une équation différentielle d'ordre deux :

$$\ddot{\theta}_i = F_i(\dot{\theta}_1, \ldots, \dot{\theta}_n, \theta_1, \ldots, \theta_n, u_1, \ldots, u_n)$$

Pour faire le lien avec l'algorithme, le système mécanique est dirigé par l'équation

différentielle suivante :

$$\dot{x}(t) = f(x,u,t) = \frac{d}{dt}\begin{pmatrix} \theta_1 \\ \dot{\theta}_1 \\ \vdots \\ \theta_n \\ \dot{\theta}_n \end{pmatrix} = \begin{cases} \dot{\theta}_1 \\ F_1(\dot{\theta}_1,\ldots,\dot{\theta}_n,\theta_1,\ldots,\theta_n,u_1,\ldots,u_n) \\ \vdots \\ \dot{\theta}_n \\ F_n(\dot{\theta}_1,\ldots,\dot{\theta}_n,\theta_1,\ldots,\theta_n,u_1,\ldots,u_n) \end{cases}$$

Notons enfin que le critère L est pré-programmé sous la forme d'expressions symboliques. L est une fonction des vitesses angulaires $\dot{\theta}_i$ et des couples u_i.

Avant de lancer l'algorithme, les diverses dérivées partielles, c'est à dire $\frac{\partial f}{\partial x}$, $\frac{\partial L}{\partial x}$, $\frac{\partial f}{\partial u}$ et $\frac{\partial L}{\partial u}$, sont construites symboliquement. Elles seront évaluées numériquement pendant l'exécution en fonction des valeurs successives du vecteur d'état et du vecteur de contrôle. Comme les valeurs calculées sont utilisées à plusieurs reprises à l'intérieur de la boucle, elles sont stockées dans des tableaux. La complexité des matrices construites est très variable. Les dérivées partielles de L forment des expressions très simples. Au contraire les dérivées partielles des fonctions F_i engendrent des expressions gigantesques (même pour un système à deux paramètres).

Exemple du pendule simple

Nous reprenons l'équation du mouvement d'un pendule simple définie au § 5.3.2. C'est une équation différentielle d'ordre deux. L'état du système x est donc composé de l'angle θ du pendule et de sa vitesse angulaire $\dot{\theta}$:

$$x^T = \left(\theta, \dot{\theta}\right)$$

L'équation du mouvement devient donc

$$\dot{x} = f(x,u,t) = \begin{cases} \dot{\theta} \\ \frac{3}{ml^2} = \frac{3}{2}\frac{g}{l}\sin\theta \end{cases}$$

Nous convenons d'autre part de minimiser une intégrale de la forme :

$$J = \int_{t_0}^{t_f} L(x,u,t)\,dt = \int_{t_0}^{t_f} (u^2 + \dot{\theta}^2)\,dt$$

qui va minimiser l'énergie dépensée (par le moteur u) et la vitesse (de manière à avoir une trajectoire lisse) au cours du mouvement. A partir de là, nous pouvons évaluer les matrices de certaines dérivées partielles nécessaires à l'application de l'algorithme exposé ci-dessus :

$$\frac{\partial f}{\partial u} = \begin{pmatrix} 0 \\ \frac{3}{ml^2} \end{pmatrix} \quad \frac{\partial f}{\partial x} = \begin{pmatrix} 0 & 1 \\ -\frac{3}{2}\frac{g}{l}\cos\theta & 0 \end{pmatrix}$$

$$\frac{\partial L}{\partial u} = (2u) \quad \frac{\partial L}{\partial x} = \begin{pmatrix} 0 \\ 2\dot{\theta} \end{pmatrix}$$

Il reste à définir les contraintes à l'instant final t_f :

$$\phi(x(t_f)) = \begin{pmatrix} \theta - \theta_f \\ \dot{\theta} - \dot{\theta}_f \end{pmatrix}$$

où θ_f et $\dot{\theta}_f$ déterminent respectivement la position et la vitesse du pendule à la fin du mouvement. Les paramètres de départ sont déterminé par les valeurs $x(t_0) = (\theta_0, \dot{\theta}_0)$.

6.5.4 Détails sur l'implémentation

Les relations entre les équations mécaniques et l'algorithme étant établies au travers de cet exemple, nous détaillons maintenant la mise en œuvre des étapes de l'algorithme. Rappelons que, même si la théorie servant de base à l'algorithme est fondée sur des équations du mouvement traitées sous leur forme continue, une étape de discrétisation est nécessaire. Ainsi, l'évolution de la fonction $u(t)$ au cours du temps est représentée par la suite des valeurs prises par cette fonction en des instants discrétisés et successifs de temps.

La première étape est une initialisation. Elle consiste à trouver une estimation des forces qui interviennent de t_0 à t_N. Meilleure est cette estimation, plus vite l'algorithme convergera. S'il est difficile de trouver des valeurs initiales de u assurant un critère réduit, en revanche, il est possible de déterminer des valeurs de u assurant le respect des contraintes. Pour cela, il suffit d'effectuer une interpolation linéaire des paramètres contenus dans le vecteur d'état. Les vitesses des paramètres sont alors constantes, leurs accélérations nulles. L'estimation de u revient à résoudre un problème de dynamique inverse. Il faut déterminer les forces permettant de réaliser cette interpolation linéaire. L'équation du mouvement est sous la forme $\ddot{q} = f(\dot{q}, q, u, t)$. Ici, $\ddot{q} = 0$ et $\dot{q}$ est une constante, il faut donc résoudre le système en u

$$f(q, u, t) = 0$$

Malheureusement, la formulation de nos équations symboliques du mouvement ne nous permet pas de résoudre ce système. Les couples et forces y figurent de manière inextricable. Par défaut, nous avons donc construit à la main les expressions des couples en fonction des paramètres θ_i et $\dot{\theta}_i$.

La deuxième étape revient à calculer le mouvement en dynamique directe à partir de la valeur courante du vecteur de contrôle. C'est, de loin, l'étape la plus coûteuse de l'algorithme. A chaque itération, il faut en effet effectuer une simulation complète du modèle mécanique. Une seconde d'animation requiert $N \times 4$ évaluations des équations F_i du mouvement, et cela pour chaque itération (il y a N intervalles de temps et l'intégration de Runge-Kutta les décompose en quatre sous-intervalles). Les résultats de l'intégration sont rangés dans les variables θ_i et $\dot{\theta}_i$. Le vecteur $\phi(t_N)$ est évalué à partir des valeurs des paramètres à l'instant final $t = t_N$. Pour cela, on effectue une différence entre les valeurs $x(t_N)$, résultats de l'intégration des équations du mouvement, et les valeurs spécifiées par l'utilisateur.

Les étapes (3) et (4) sont effectuées simultanément. Les expressions symboliques des dérivées partielles $\frac{\partial f}{\partial x}$, $\frac{\partial L}{\partial x}$ sont évaluées numériquement à chaque pas de l'intégration (quatre fois par intervalle de temps). L'étape (3) consiste à intégrer les deux équations différentielles décrivant les variations du vecteur p et de la matrice R à partir des valeurs successives des dérivées partielles. L'intégration se fait cette fois en marche arrière, en allant de t_N vers t_0. Les résultats $p(t_i)$ et $R(t_i)$ sont stockés pour être utilisés à la dernière étape (on ne garde pas les résultats intermédiaires utilisés par la méthode d'intégration). Les intégrales de l'étape (4) sont calculées en parallèle, de l'arrière vers l'avant, par une simple sommation. Par exemple, l'intégrale A est estimée de la manière suivante :

$$A = \frac{1}{N+1} \sum_{t_0}^{t_N} R^T \frac{\partial f}{\partial u} \left(\frac{\partial f}{\partial u} \right)^T R$$

On ne calcule les dérivées partielles $\frac{\partial f}{\partial u}$ et $\frac{\partial L}{\partial u}$ qu'à chaque intervalle de temps. Par contre, celles-ci sont stockées dans deux tableaux de matrices pour être utilisées à l'étape (7).

Les étapes (5) et (6) ne posent pas de problèmes particuliers. Ici, l'inversion de A est réalisée numériquement. Précisons que nous utilisons un paramètre $\epsilon = 0.1$. Une valeur plus grande entraine souvent des divergences.

La dernière étape consiste à mettre à jour le vecteur de contrôle. On utilise les valeurs numériques – calculées auparavant – des dérivées partielles par rapport à u et des matrices p et R.

Les conditions d'arrêt sont évaluées par un calcul matriciel

6.5.5 Résultats

L'algorithme qui précède a été utilisé pour animer un bras articulé disposant de deux articulations (figure 6.2). Le premier élément est caractérisé par sa masse M_1 et sa longueur L_1, le deuxième par M_2 et L_2. L'état du système est donc composé des angles θ_1 et θ_2 ainsi que des vitesses angulaires $\dot{\theta}_1$ et $\dot{\theta}_2$:

$$x^T = \left(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2 \right)^T$$

Les conditions initiales sont $\theta_1 = \theta_2 = 0$ et $\dot{\theta}_1 = \dot{\theta}_2 = 0$. Les conditions finales sont $\theta_1 = \frac{1}{2}$, $\theta_2 = \frac{3}{2}$ et $\dot{\theta}_1 = \dot{\theta}_2 = 0$. Le vecteur de contrôle est constitué des couples u_1 et u_2 appliqués à chaque élément du bras :

$$u^T = (u_1, u_2)$$

et le critère est de la forme :

$$L = \alpha_1 u_1^2 + \alpha_2 u_2^2 + \dot{\theta}_1^2 + \dot{\theta}_2^2$$

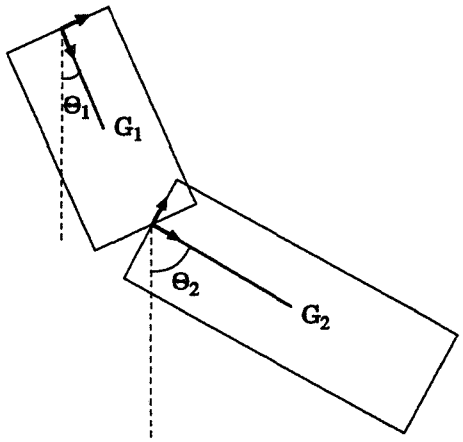


FIG. 6.2 Description du pendule double.

Chaque figure est constituée de trois graphiques :

- Sur la gauche sont affichées les valeurs des couples au cours du temps. u_1 est représenté par le symbole ▪ , u_2 par le symbole △ . Le contrôle total (en valeur absolue) consommé par le mouvement est affiché en haut à droite.
- Au milieu figure l'évolution du vecteur d'état :
  - θ_1 est représenté par ▪ ,
  - θ_2 par △ ,
  - $\dot{\theta}_1$ par ×,
  - $\dot{\theta}_2$ par ◇.
- La partie droite représente l'animation du bras

D'une figure à l'autre, l'échelle varie. Elle est représentée sur la gauche des graphiques par les valeurs extrêmes que prennent les paramètres. La ligne horizontale indique une valeur nulle.

Les paramètres que nous avons fait varier sont les masses M_1 et M_2 et les coefficients α_1 et α_2 affectant le critère à minimiser. Les longueurs sont fixes et égales à un. La durée de l'animation est d'une seconde.

La figure (6.3) sert de point de comparaison aux suivantes. D'abord, on peut constater que les contraintes finales sont effectivement respectées. Sur la partie gauche, on remarque que le premier élément consomme moins d'énergie que le second, bien qu'il supporte les deux masses. L'explication vient du fait que la deuxième articulation subit une rotation trois fois plus importante. A la fin du mouvement, u_2 agit comme un frein pour compenser la vitesse acquise et rétablir la contrainte $\dot{\theta}_2 = 0$. Au contraire, le premier élément ne freine presque pas puisqu'il est ralenti par la masse des deux bras.

Quand la masse du premier bras est augmentée (figure 6.4), le bras inférieur commence par tourner dans le sens inverse. L'interprétation est, peut-être, qu'il aide le premier à atteindre son objectif. Par effet de réaction, le deuxième bras augmente l'inertie du bras supérieur.

Sur la figure (6.5), le phénomème est inversé. C'est la masse du deuxième bras qui est augmentée et c'est le bras supérieur qui commence son mouvement à contre-sens. La prise d'élan se situe à un endroit où la pesanteur n'exerce presque pas de couple. Dans la deuxième partie du mouvement, quand l'effet du poids est à son maximum, la rotation est très rapide afin de le subir le moins longtemps possible (on peut faire une analogie avec le lever d'une barre par un haltérophile). Le pendule à une connaissance globale du mouvement et peut ainsi anticiper.

Les deux figures suivantes illustrent l'influence du critère sur le mouvement. Sur la première (figure 6.6), le coefficient affecté au contrôle u_1 a été augmenté. C'est, toujours par effet de réaction, le deuxième bras qui prend en charge le mouvement. Le phénomène inverse se produit sur la figure (6.7) où c'est le deuxième couple qui est minimisé.

Enfin, à titre de comparaison, la figure (6.8) montre les couples utilisés pour réaliser le mouvement par interpolation linéaire.

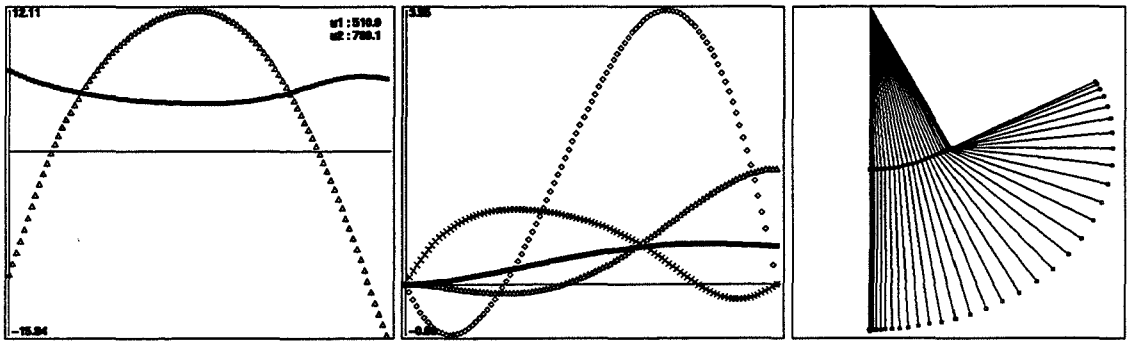


FIG. 6.3 $M_1 = 1, M_2 = 1, \alpha_1 = 1, \alpha_2 = 1$

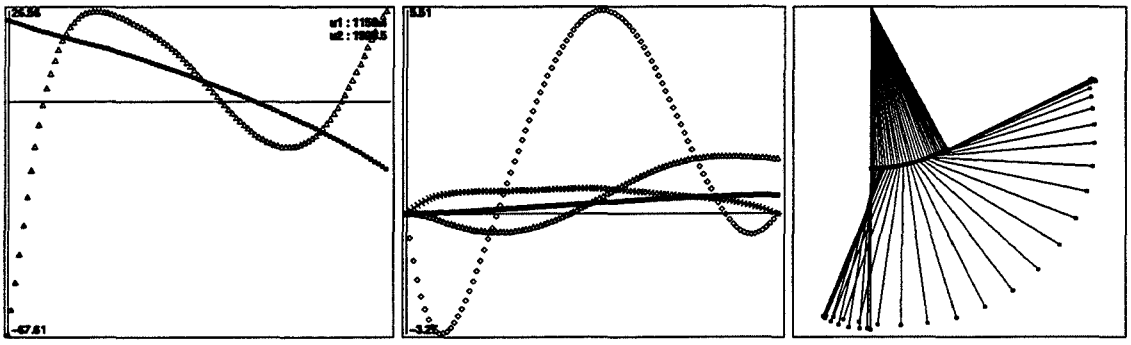


FIG. 6.4 $M_1 = 10, M_2 = 1, \alpha_1 = 1, \alpha_2 = 1$

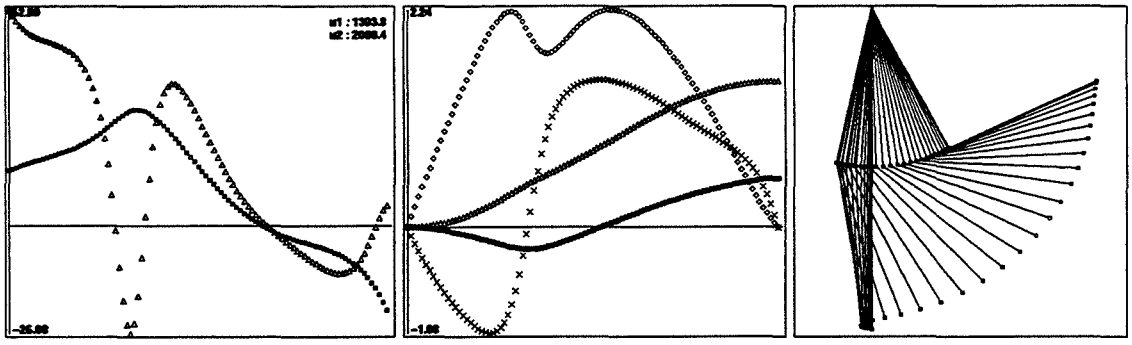


FIG. 6.5 $M_1 = 1, M_2 = 10, \alpha_1 = 1, \alpha_2 = 1$

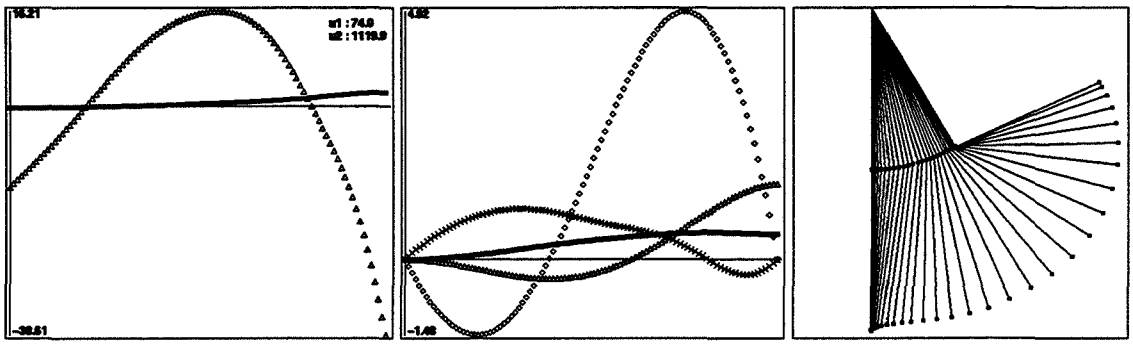


FIG. 6.6 $M_1 = 1$, $M_2 = 1$, $\alpha_1 = 10$, $\alpha_2 = 1$

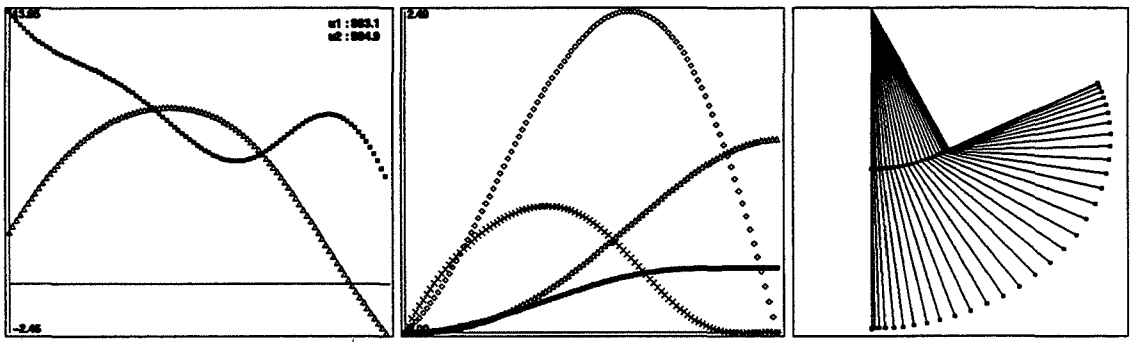


FIG. 6.7 $M_1 = 1$, $M_2 = 1$, $\alpha_1 = 1$, $\alpha_2 = 10$

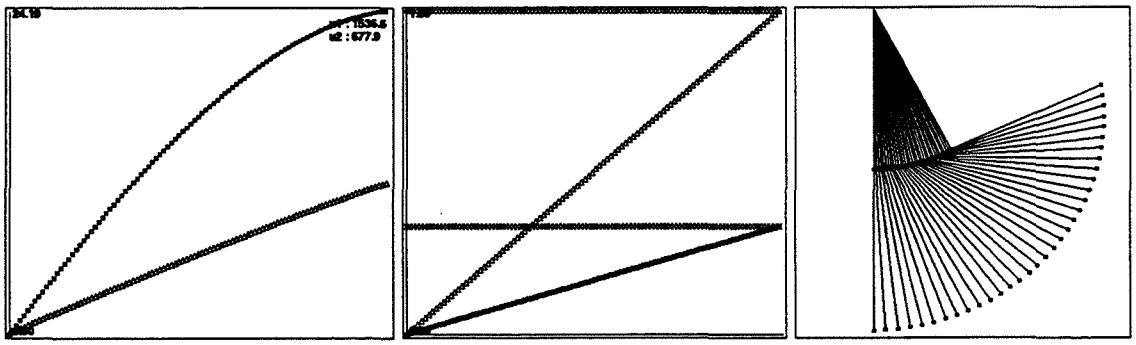


FIG. 6.8 Interpolation linéaire

Discussion

Nous avons exposé diverses méthodes basées sur les techniques d'optimisation ou du contrôle optimal. On peut en dégager un certain nombre de points communs.

D'abord, toutes permettent une coopération entre modèle descriptif et modèle générateur en ce sens que le mouvement respecte les lois de la dynamique et que des contraintes sont posées par l'animateur non seulement au début, mais aussi au cours du mouvement. En fait, l'introduction de ces méthodes permet de définir une nouvelle race d'interpolateurs. Les méthodes classiques d'interpolation (telles que les courbes splines) ne prennent en compte que la cinématique des objets animés. De plus, l'animateur est obligé de spécifier explicitement certains paramètres tels que l'abscisse curviligne le long de la courbe d'interpolation. La mise au point d'une animation nécessite de longs et fastidieux essais.

Au contraire, les interpolateurs dont le principe est la minimisation de l'énergie dépensée au cours du mouvement tiennent compte de l'aspect mécanique des objets. De la sorte, ce n'est pas à l'animateur de simuler le réalisme du mouvement; le programme s'en charge lui-même. La connaissance du mouvement global, par l'intermédiaires des contraintes finales, permet d'obtenir des effets d'anticipation – par exemple, la prise d'élan – très réalistes. Cette connaissance du futur peut néanmoins parfois être un inconvénient. Par exemple, si l'on veut simuler une poursuite, le chasseur connait la position finale de sa cible. Il n'a qu'à se rendre en ligne droite vers cette position pour y attendre sa proie.

Le rôle de l'animateur n'est pas négligeable. En effet, l'allure du mouvement est grandement influencée par les caractéristiques mécaniques et le type de la fonction critère. Nous avons vu que, même pour un bras à deux degrés de liberté, le choix de ces différents paramètres influence profondemment l'animation. Le choix du critère reste encore un problème à résoudre. Faut-il minimiser l'énergie au dépend d'une trajectoire plus lisse? Parmi les différentes forces entrant en jeu, lesquelles minimiser en priorité?

En ce qui concerne l'implantation des différentes approches, des similitudes apparaissent aussi. Premièrement, il est indispensable de disposer des équations symboliques du mouvement (excepté pour la dernière méthode évoquée). En effet, que ce soit pour l'optimisation ou le contrôle optimal, il est nécessaire de calculer des dérivées partielles, des gradients,... Cette contrainte restreint le choix du modèle mécanique. Nous avons utilisé le formalisme de Wittenburg mais celui de Lagrange est aussi bien adapté pour la méthode d'optimisation, puisque les équations du mouvement sont utilisées comme des contraintes implicites.

Une autre constante est la complexité en temps et place requise par ces méthodes. En place d'abord, puisque que le modèle mécanique (ainsi que les dérivées partielles) requiert une représentation symbolique encombrante et dont les besoins en espace augmentent dramatiquement avec la complexité des objets à animer. Par rapport à la simulation où seul le nombre de degrés de liberté interviennent dans la complexité, ici la complexité de l'animation dépend aussi de la durée. De plus, l'animation est considérée dans sa globalité et non pas instant par instant comme en dynamique directe. Il est donc indispensable de conserver en permanence des informations concernant l'état du système à chaque instant de l'animation. L'échantillonnage temporel doit être déterminé a priori ce qui requiert de spécifier un intervalle de temps suffisamment petit pour éviter des divergences dues à l'instabilité des équations mécaniques.

Au niveau du temps d'exécution et par rapport à l'animation résolvant le mouvement par dynamique directe, ces méthodes demandent aussi un saut en complexité. En effet, elles sont basées sur des algorithmes itératifs. A chaque itération, il est nécessaire, entre autres, de calculer le mouvement en dynamique directe du système. L'utilisation de ces techniques comme des interpolateurs que l'on pourrait manipuler interactivement comme des splines semble donc pour l'instant exclue.

Un point commun à toutes ces méthodes est que la solution atteinte n'est en général pas le minimum absolu mais un minimum local. D'où l'importance de lancer l'algorithme avec des conditions initiales les plus proches possibles d'un minimum local acceptable. Ici, le terme "conditions initiales" signifie non pas état du système à l'instant initial mais bien une description estimée de l'animation à chaque intervalle de temps. La solution couramment employée consiste à réaliser cette estimation à l'aide d'une interpolation par courbes splines.

Si toutes les méthodes qui font l'objet de ce rapport ont un principe commun, il n'en reste pas moins que la forme change. Quatre techniques distinctes ont été présentées:

1 optimisation non linéaire

2 programmation dynamique

3 contrôle optimal pour des équations linéaires avec un critère quadratique

4 contrôle optimal pour des équations non linéaires avec un critère quelconque

De par le principe même de la programmation dynamique (une complexité croissant exponentiellement avec la taille des données manipulées), la première solution ne semble envisageable que pour des objets simples, et qui plus est, pour des simulations de courte durée. Dans un papier ultérieur [vdPFV92], les auteurs proposent de n'utiliser leur méthode que pour calculer le mouvement des parties les plus significatives d'un solide complexe. Le mouvement des autres éléments est déterminé par des solutions plus rapides telles que la dynamique directe ou la dynamique contrainte. De cette manière, ils arrivent à animer un modèle humain s'adaptant automatiquement à la forme du sol. Seul le balancement des jambes est contrôlé par programmation dynamique.

La seconde approche est, elle aussi, très limitée puisque rares sont les mouvements qui peuvent être décrits au moyen d'équations linéaires. D'ailleurs, l'exemple fourni par les auteurs est très simple. L'intérêt principal est la solution proposée pour assurer la continuité du contrôle entre plusieurs mouvements successifs. Elle pourrait être adaptée à notre solution.

La solution que nous proposons prend en compte les équations non linéaires au prix d'un algorithme plus coûteux. C'est une méthode du premier ordre, donc relativement lente à converger. L'utilisation d'une méthode d'ordre deux, en particulier à l'approche de l'optimum, serait souhaitable mais au prix d'une complexité encore accrue.

Enfin, la denière solution, fondée sur une discrétisation du mouvement, semble offrir le plus de possibilités et de souplesse. En particulier, il est très facile d'ajouter des contraintes quelconques à n'importe quel moment de l'animation. Par contre, l'algorithme est très gourmand puisque tous les paramètres, à chaque intervalle de temps, doivent être intégrés dans une seule et gigantesque matrice qui doit être inversée à chaque itération. En comparaison, notre solution utilise beaucoup plus de matrices (de l'ordre du nombre d'intervalles de temps) mais de tailles réduites (de l'ordre du nombre de degrés de libertés) ce qui, globalement, est plus économique.

D'importants développements semblent encore nécessaires. Par exemple la prise en compte de contraintes à base d'inéquations. Il serait aussi souhaitable de limiter l'amplitude du contrôle (forces et couples) utilisé, de manière à rester dans les limites du réel. L'approche de la programmation dynamique est la seule à pouvoir tenir compte de ce genre de contraintes, puisque tant les degrés de liberté que les forces sont représentés par un ensemble de valeurs bornées.

Avec l'augmentation permanente de la puissance des machines, l'utilisation des méthodes présentées devrait se répandre. Ce sont cependant des solutions demandant d'importants développements, et en particulier :

- un module de dynamique générant des équations symboliques du mouvement,
- des bibliothèques de calcul matriciel et symbolique,
- un module de calcul numérique (intégration de systèmes d'équations différentielles, calcul d'intégrales,...),
- un algorithme d'optimisation ou de contrôle optimal.

Enfin, il reste à rendre ces techniques accessibles à un utilisateur non informaticien. Pour l'instant, la plupart des données (contraintes, type de critère) doivent être préprogrammées. Il semble possible de créer une interface habituelle (type positions clés) dont l'interpolateur serait basé sur une des méthodes présentées dans ce rapport.

En résolvant certains des problèmes énoncés plus haut, un papier récent [Coh92], qui étend la méthode de Witkin et Kass, laisse présager de l'importance accrue des méthodes d'optimisation. L'une des clés repose sur la décomposition du problème global en un ensemble de problèmes plus petits et bien plus simples à résoudre. L'interface avec l'utilisa-

teur est aussi améliorée. En particulier, celui-ci peut aider le système à trouver la solution adéquate (résolution des problèmes de minimum locaux).

Conclusion Générale

Les chapitres de la première partie constituent l'intérêt essentiel de cette thèse. Les travaux qui y sont présentés sont partis de l'idée d'appliquer les techniques de manipulation de courbes paramétriques relatives aux positions aux cas des orientations Nous voulions avoir sur ces dernières des méthodes de contrôle – en particulier interactives – aussi souples que celles existant dans $\mathcal{R}^3$. Notre modèle de référence a été les polynômes d'interpolation d'Hermite, qui par la manipulation des tangentes à la courbe, permet d'offrir à l'utilisateur des techniques de contrôle efficaces, intuitives et qui plus est, faciles à implanter.

La définition de la notion de tangente sphérique aux courbes interpolant des orientations à permis d'atteindre cet objectif. Il est ainsi possible, par l'intermédiaire desdites tangentes de générer des effets de tension, de discontinuité ou de biais sur des trajectoires d'orientation; et ceci de façon analogue à ce qui se pratique dans $\mathcal{R}^3$. L'application quasi-immédiate de l'interpolation d'Hermite à l'espace des orientations s'est révélée possible au moyen d'une paramétrisation simple des orientations par les logarithmes de quaternion.

En contre-partie d'une grande souplesse d'utilisation, nous avons montré que les trajectoires constituées de morceaux de splines cubiques posent des problèmes au niveau de la cinétique du mouvement. La vitesse du déplacement le long de telles trajectoires dépend aussi bien de la tension des tangentes que de l'espacement entre les points de contrôle. Pour y remédier, un algorithme de reparamétrisation, conservant la trajectoire spatiale initiale mais en affectant la cinétique, a été présenté. Inspiré de lois élémentaires de la mécanique, il permet d'obtenir des mouvements plus réalistes; et cela sans intervention de l'utilisateur. C'est spécialement intéressant dans le cas des orientations. La cinétique du mouvement peut néanmoins être controlée en imposant des vitesses – linéaires ou angulaires – en certains points de la trajectoire. Enfin, cette méthode peut aussi servir à définir une courbe de reparamétrage que l'animateur aura le loisir d'utiliser afin d'ajuster plus finement et selon son inspiration la cinétique du mouvement.

De manière à gérer aisément les interactions entre les applications décrites ci-dessus et l'utilisateur, un interface 3-D de haut niveau a été proposé. Sa capacité à mettre en relation directe les mouvements tri-dimensionnels des divers éléments composant la scène avec ceux de la souris en font un outil très intuitif. L'intégration dans une application interactive 3-D est rapide. C'est à la charge de cette dernière de gérer la définition et la sélection des sous-espaces contraignant les mouvements à ne dépendre que de deux degrés de liberté; l'interface se charge de faire correspondre la souris et ces sous-espaces. Cette obligation permet d'utiliser l'interface 3-D dans des applications très générales. Ainsi, la

gestion des orientations, positions et tangentes dans notre modeleur d'animation se fait très naturellement à l'aide de ces fonctionnalités.

Concernant l'interaction, nous avons aussi proposé un algorithme autorisant la manipulation interactive d'un solide articulé. La méthode n'offre pas toutes les fonctionnalités couramment disponibles dans la plupart des algorithmes classiques de cinématique inverse. En particulier, les contraintes angulaires ne sont pas prises en compte. Les limitations sont dues à la paramétrisation exclusive du solide articulé par les positions des différents élément dont il est constitué – et non par leurs orientations comme cela a toujours été fait. Mais c'est ce choix qui fait la force de cet algorithme. N'utilisant que des opérations élémetaires sur des vecteurs, il est rapide mais aussi robuste et très simple à implanter, au contraire des autres méthodes qui utilisent des algorithmes numériques souvent instables et lents. Son emploi se justifie dans un contexte interactif.

C'est d'ailleurs la caractéristique de toutes ces fonctionnalités d'être suffisamment simples et performantes pour être utilisées avec profit dans un environnement où le dialogue avec l'animateur prime. A notre avis, l'intérêt principal réside dans la possibilité de pouvoir manipuler sur une même zone de dialogue les caractéristiques fondamentales d'une animation. Non seulement l'animateur n'a pas à faire la navette entre une pléthore de fenêtres graphiques, mais surtout, cette concentration des informations lui permet de mieux appréhender les relations entre tous les paramètres qui constituent une animation.

Ces techniques propres à l'animation ont aussi été appliquées à la modélisation interactive des cylindres généralisés. Les règles de balayage concernant les rotations traditionnellement limitées à un degré de liberté sont étendues à des rotations quelconques. Les cas pratiquement inévitables d'auto-intersection sont résolus en utilisant les opérations booléennes des arbres de construction. La méthode est brutale mais a l'avantage de bien fonctionner. Enfin, l'algorithme de reparamétrisation offre indirectement une solution à la facettisation adaptive lors de la détermination des frontières du cylindre : les facettes sont plus concentrées dans les zones présentant une forte courbure.

Une partie des résultats de cette première partie sont présentés dans [HP93].

Dans la deuxième partie, nous nous sommes intéressés à des méthodes d'interpolation de haut niveau. Trois techniques basées sur des méthodes numériques de minimisation ont été détaillées. Nous y avons présenté une alternative, basée sur le contrôle optimal des équations continues du mouvement. Les résultats obtenus, en comparaison de la somme de travail fournie (construction symbolique des équations du mouvement, algorithme de contrôle optimal), peuvent sembler dérisoires. Les méthodes d'optimisation nous semblent cependant promises à un bel avenir et nous espérons en avoir montré l'intérêt sans en cacher les aspects contraignants. Elles allierons les avantages des systèmes descriptifs et des systèmes basés sur un modèle réel. Des premiers, on peut tirer parti des possibilités de contrôle des trajectoires; des seconds, du réalisme du mouvement généré.

On n'en est pas encore à utiliser les méthodes d'optimisation comme des interpolateurs ainsi qu'on le fait aujourd'hui avec les splines. En particulier, se pose le problème

de choisir un critère adéquat afin d'obtenir un mouvement spécifique. En plus, ces techniques font une consommation gargantuesque d'espace mémoire et leur lenteur interdit pour l'instant toute application interactive. A l'instar de ce que propose Van de Panne, un des remèdes serait de décomposer le gigantesque problème initial en sous-problèmes, beaucoup plus simples; certains d'entre eux pouvant même être résolus par d'autre méthodes que l'optimisation, cette dernière étant principalement consacrée à l'interpolation des paramètres déterminants de l'animation. A en juger par l'utilisation sans cesse croissante des méthodes d'optimisation, et pas seulement en animation, des développements futurs devraient améliorer leur efficacité et leur rapidité.

Pourquoi ne pas rêver, dans un avenir peut-être proche, à un modeleur hybride intégrant les fonctionnalités des modèles descriptifs et générateurs.

Annexe A

Quaternions

On donne dans cet appendice un ensemble de démonstrations concernant les quaternions introduits dans le premier chapitre, puis une liste de fonctions permettant la manipulation des quaternions.

A.1 Démonstrations

A.1.1 Produit de deux quaternions et construction du quaternion représentant une orientation d'angle θ

Nous montrons ici pourquoi une rotation d'angle θ autour d'un axe v est représentée par le quaternion

$$q = \left[\cos\frac{\theta}{2}, \sin\frac{\theta}{2}v\right]$$

et l'explication de la formule permettant d'appliquer une telle rotation à un vecteur u.

Nous étudierons le cas où u et la partie vectorielle du quaternion $q = [w, v]$ sont orthogonaux puis celui où ils sont colinéaires. L'examen des deux résultats permettra de tirer une règle générale sur la rotation d'un vecteur par un quaternion. Dans les calculs qui suivent, le vecteur u est traité comme un quaternion de partie réelle nulle : $u = [0, u]$.

Quand u et v sont perpendiculaires, le produit qu est donné par :

$$\begin{aligned} qu &= [w, v][0, u] \\ &= [-vu, wu + v \wedge u] \\ &= [0, wu + v \wedge u] \end{aligned}$$

Comme la partie réelle est nulle, le résultat représente un nouveau vecteur qui est le vecteur u tourné autour de v (figure A.1). Pour déterminer les caractéristiques de ce vecteur, déterminons sa longueur et la mesure de l'angle dont il a été tourné.

Le carré de la longueur de ce nouveau vecteur est :

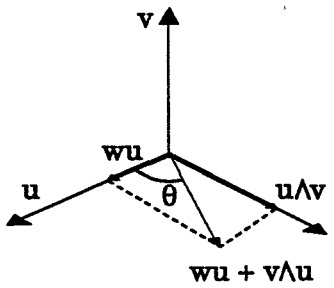


FIG. A.1 L'axe de rotation v et le vecteur u sont orthogonaux.

$$\begin{aligned} \|wu + v \wedge u\|^2 &= \|wu\|^2 + \|v \wedge u\|^2 \\ &= w^2\|u\|^2 + \|v\|^2\|u\|^2 \\ &= (w^2 + \|v\|^2)\,\|u\|^2 \\ &= \|u\|^2 \end{aligned}$$

La longueur du vecteur résultat est donc la même que celle du vecteur original.

L'angle de rotation est calculé à partir des longueurs $\|wv\|$ et $\|wu + v \wedge u\| = \|u\|$:

$$\cos\theta = \frac{\|wu\|}{\|u\|} = w$$

et comme q est un quaternion unitaire,

$$\begin{aligned} w^2 + v^2 &= 1 \\ \cos^2\theta + v^2 &= 1 \end{aligned}$$

d'où

$$\|v\| = \sin\theta$$

La multiplication par un quaternion unitaire provoque une rotation d'angle θ quand le vecteur auquel elle est appliquée est perpendiculaire à la partie réelle du quaternion.

Considérons maintenant le cas où u est colinéaire à v (c'est à dire $\alpha u = v$. La rotation d'un vecteur qui se trouve sur l'axe de rotation devrait laisser ce vecteur inchangé. Pour le vérifier, multiplions à nouveau u par q :

$$\begin{aligned} qu &= [w, v][0, u] \\ &= [-vu, wu + v \wedge u] \\ &= [-vu, wu] \end{aligned}$$

Comme la partie scalaire du résultat est non nulle, qu ne représente pas une vraie rotation. Pour obtenir un quaternion de partie scalaire nulle, examinons le double produit quq^{-1} :

$$\begin{aligned} quq^{-1} &= [-vu, wu][w, -v] \\ &= [0, (vu)v + w^2u - wu \wedge v] \\ &= [0, (vu)v + w^2u] \\ &= [0, (\alpha uu)\alpha u + w^2u] \\ &= [0, (\alpha u \alpha u)u + w^2u] \\ &= [0, v^2u + w^2u] \\ &= [0, u] \end{aligned}$$

ce qui montre que u est inchangé.

Revenons maintenant au cas perpendiculaire et montrons que le produit uq^{-1} est équivalent au produit qu :

$$\begin{aligned} uq^{-1} &= [0, u][w, -v] \\ &= [uv, uw - u \wedge v] \\ &= [0, uw + v \wedge u] \\ &= [0, qu] \end{aligned}$$

quq^{-1} applique donc une rotation qui est le double de celle engendrée par qu autour du même axe, ce qui explique l'apparition du terme $\frac{\theta}{2}$ dans l'expression d'un quaternion représentant une rotation d'angle θ. quq^{-1} est la formule générale pour appliquer une rotation à un vecteur u.

A.1.2 Elévation d'un quaternion à la puissance n

Soit

$$q = \left[\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\right] \quad \|v\| = 1$$

Montrons par récurrence que

$$q^n = \left[\cos\frac{\theta n}{2}, \sin\frac{\theta n}{2}\right]$$

On a donc

$$q(n-1) = \left[\cos\frac{\theta(n-1)}{2}, \sin\frac{\theta(n-1)}{2}\right]$$

d'où

$$\begin{aligned} q^n &= q(n-1)q \\ &= \left[\cos\frac{\theta(n-1)}{2}, \sin\frac{\theta(n-1)}{2}\right]\left[\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\right] \\ &= \left[\cos\frac{\theta(n-1)}{2}\cos\frac{\theta}{2} - \sin\frac{\theta(n-1)}{2}\sin\frac{\theta}{2}vv, \cos\frac{\theta(n-1)}{2}\sin\frac{\theta}{2}v + \sin\frac{\theta(n-1)}{2}\cos\frac{\theta}{2}v + v \wedge v\right] \end{aligned}$$

$$= \left[\cos\left(\frac{\theta(n-1)}{2}+\frac{\theta}{2}\right), \sin\left(\frac{\theta(n-1)}{2}+\frac{\theta}{2}\right) v\right]$$
$$= \left[\cos\frac{\theta n}{2}, \sin\frac{\theta n}{2}\right]$$

A.1.3 Interpolation linéaire de deux quaternions

La formule d'interpolation linéaire $slerp(q_1, q_2, u)$ entre deux quaternions q_1 et q_2, à partir des propriétés du groupe est obtenue en appliquant à l'orientation q_1 une fraction u de la rotation conduisant q_1 en q_2.

La rotation transformant l'orientation q_1 en l'orientation q_2 est représentée par le quaternion : $q_1^{-1}q^2$.

Une fraction u de cette rotation est obtenue en lui appliquant l'élévation à la puissance u : $(q_1^{-1}q^2)^u$

En appliquant cette dernière rotation à l'orientation q_1 de départ, on trouve la formule :

$$slerp(q_1, q_2, u) = q_1 \left(q_1^{-1}q_2\right)^u$$

A.1.4 Logarithme d'un quaternion

Si $q = [w, v]$ est un quaternion unitaire, alors le logarithme $[0, V] = \log q$ de ce quaternion est défini à partir de l'exponentielle des quaternions de partie réelle nulle par :

$$\exp([0, V]) = q$$

Comme

$$\exp([0, V]) = \left[\cos\|V\|, sin\|V\|\frac{V}{\|V\|}\right]$$

nous avons :

$$\begin{cases} w &= cos\|V\| \\ v &= \frac{V}{\|V\|}\sin\|V\| \end{cases}$$

$$\begin{cases} \|V\| = \cos^{-1} w \\ V = \frac{v\|V\|}{\sin\|V\|} \end{cases}$$

Quand $w \neq 1$,

$$V = \frac{v\|V\|}{\sqrt{1-\cos^2\|V\|}} = \frac{\cos^{-1} w}{\sqrt{1-w^2}}v$$

ce qui, en récapitulant, donne :

$$\log q = \begin{cases} \frac{\cos^{-1} w}{\sqrt{1-w^2}}v & \text{si} \quad w \neq 1 \\ 0 & \text{sinon} \end{cases}$$

A.2 Librairie de fonctions C

```
#define X 0
#define Y 1
#define Z 2
#define W 3

typedef double[4] Quaternion;

typedef double[3] Vecteur;
```

- Initialisation d'un quaternion de norme quelconque

```
void Q_Init( q, w, x, y, z)
Quaternion q;
  double w, x, y, z;
{
  q[W] = w;
  q[X] = x;
  q[Y] = y;
  q[Z] = z;
}
```

- Initialisation d'un quaternion de partie réelle nulle à partir d'un vecteur

```
void Q_Vecteur( q, v)
Quaternion q;
  double x, y, z;
{
  q[W] = 0.0;
  q[X] = v[X];
  q[Y] = v[Y];
  q[Z] = v[Z];
}
```

- Initialise un quaternion par la donnée d'un angle (en radians) et d'un axe de rotation v. Le quaternion résultant est unitaire.

```
void Q_Rotation( q, angle, x, y, z)
  Quaternion q;
  Vecteur v;
  double angle;
```

```
{
  double norme = V_Norme( v);
         sn = sin( angle/2.);

  q[W] = cos( angle/2.);
  q[X] = sn * x / norme;
  q[Y] = sn * y / norme;
  q[Z] = sn * z / norme;
}
```

- Initialise un quaternion par la donnée des angles d'Euler (en radians). Cette rotation est la composée de trois rotation. Une première d'un angle a autour de l'axe des z, une deuxième d'un angle b autour de l'axe des x et enfin une dernière rotation d'un angle c, à nouveau autour de l'axe des z.

```
void Q_Euler( q, a, b, c)
Quaternion q;
  double a, b, c;
{
  double ca, sa,
         cb, sb,
         cc, sc;

  /* demi-angle en radians*/
  a /= 2.;
  b /= 2.;
  c /= 2.;

  ca = cos( a); sa = sin( a);
  cb = cos( b); sb = sin( b);
  cc = cos( c); sc = sin( c);

  q[W] = ca*cb*cc - sa*cb*sc;
  q[X] = ca*sb*cc + sa*sb*sc;
  q[Y] = ca*sb*sc - sa*sb*cc;
  q[Z] = sa*cb*cc + ca*cb*sc;
  }
```

- Produit des deux quaternions $q_1 = (w_1, v_1)$ et $q_2 = (w_2, v_2)$. Le quaternion résultat q peut être la même variable que q_1 ou q_2.

$$(w1, v1).(w1, v2) = (w1.w2 - v1.v2, t1.v2 + t2.v1 + v1xv2)$$

```
void Q_Produit( q1, q2, q)
```

```
  Quaternion q1, q2, q;
{
  Quaternion qq;

  qq[W] = q1[W]*q2[W] - ( q1[X]*q2[X]+q1[Y]*q2[Y]+q1[Z]*q2[Z] ) ;
  qq[X] = q1[W]*q2[X]+q2[W]*q1[X]+q1[Y]*q2[Z]-q1[Z]*q2[Y];
  qq[Y] = q1[W]*q2[Y]+q2[W]*q1[Y]+q1[Z]*q2[X]-q1[X]*q2[Z];
  qq[Z] = q1[W]*q2[Z]+q2[W]*q1[Z]+q1[X]*q2[Y]-q1[Y]*q2[X];
  q[W] = qq[W];
  q[X] = qq[X];
  q[Y] = qq[Y];
  q[Z] = qq[Z];
}
```

- Applique une rotation d'un quaternion q à un vecteur v_1. Cette fonction effectue le double produit :

$$(0, v_2) = q(0, v_1)q^{-1}$$

```
void Q_Applique( q, v1, v2)
Quaternion q;
Vecteur v1, v2;
{
  Quaternion qq;

  /*
  qq = q.(0,v)
  */
  qq[W] = - ( q[X]*v1[0] + q[Y]*v1[1] + q[Z]*v1[2] );
  qq[X] = q[W] * v1[0] + q[Y] * v1[2] - q[Z] * v1[1];
  qq[Y] = q[W] * v1[1] + q[Z] * v1[0] - q[X] * v1[2];
  qq[Z] = q[W] * v1[2] + q[X] * v1[1] - q[Y] * v1[0];

  /*          -1
  v2 = img( qq  .q )
  */
  v2[0] = - qq[W]*q[X] + q[W]*qq[X] - qq[Y]*q[Z] + qq[Z]*q[Y];
  v2[1] = - qq[W]*q[Y] + q[W]*qq[Y] - qq[Z]*q[X] + qq[X]*q[Z];
  v2[2] = - qq[W]*q[Z] + q[W]*qq[Z] - qq[X]*q[Y] + qq[Y]*q[X];
}
```

- Détermine le quaternion inverse q^{-1} de q.

```
void Q_Inverse( q, qi)
  Quaternion q, qi;
```

```
{
  qi[W] = q[W];
  qi[X] = -q[X];
  qi[Y] = -q[Y];
  qi[Z] = -q[Z];
}
```

- Calcule l'exponentielle d'un quaternion. En fait, la partie réelle de ce quaternion doit être nulle ce qui explique que l'argument est un vecteur. Le résultat est un quaternion.

```
Q_Exp( v, q)
Vecteur v;
Quaternion q;
{
  double norme = V_Norme( v);
  double sn;

  if ( norme <  epsilon ) {
    Q_Init( q, 1., 0., 0., 0.);
  }
  else {
    sn = sin( norme)/norme;
    q[W] = cos( norme);
    q[X] = sn*v[0];
    q[Y] = sn*v[1];
    q[Z] = sn*v[2];
  }
}
```

- Calcule le logarithme d'un quaternion. Le résultat est un quaternion de partie réelle nulle, i.e. un vecteur.

```
void Q_Log( q, v)
  Vecteur v;
  Quaternion q;
{
  double norme = V_Norme( q);
  double angle = acos( q[W]);
  int i;

  if ( norme < epsilon ) {
    for ( i = X ; i <= Z ; i++ ) v[i] = 0.0;
  } else {
    angle /= norme;
```

```
    for ( i = X ; i <= Z ; i++ )
      v[i] = angle * q[i];
  }
}
```

- Produit scalaire de deux quaternions. Ce produit scalaire représente le cosinus de l'angle dans $\mathcal{R}^4$ entre les deux quaternions.

```
double Q_ProduitScalaire( q0, q1)
  Quaternion q0, q1;
{
  return q0[W]*q1[W] + q0[X]*q1[X] + q0[Y]*q1[Y] + q0[Z]*q1[Z];
}
```

- Evalue le quaternion q interpolant q_1 et q_2 à partir du paramètre *alpha* ($0 \leq alpha \leq 1$).

```
void Q_Interpole( q1, q2, alpha, q)
  Quaternion q1, q2;
  double alpha;
  Quaternion q; /* resultat */
{
  double angle = acos( Q_ProduitScalaire( q1, q2));
  double beta1, beta2;
  int i;

  if ( angle < epsilon ) {
    beta1 = 1. - alpha; beta2 = alpha;
  } else {
    beta1 = sin( ( 1. - alpha ) * angle ) / sin( angle);
    beta2 = sin( alpha * angle ) / sin( angle );
  }
  for ( i = X ; i <= W ; i++ )
    q[i] = beta1 * q1[i] + beta2 * q2[i];
}
```

- Extrait l'axe de rotation (normé) d'un quaternion

```
void Q_AxeDeRotation( q, v)
  Quaternion q;
  Vecteur v;
{
  /* la norme de la partie imaginaire du quaternion */
  double norme = V_Norme( q);
```

```
  int i;

  for ( i = X ; i <= Z ; i++ ) v[i] = q[i]/norme;
}
```

- Extrait l'angle de rotation d'un quaternion

```
double Q_AngleDeRotation( q)
  Quaternion q;
{
  return 2*acos( q[W]);
}
```

- Détermine le quaternion q permettant de passer du repère de référence à un repère défini par le trièdre (a, b, c) (voir § 4.5)

```
void Q_ChangementRepere( a, b, c, q)
  Quaternion q;
  Vecteur a, b, c;
{
  double aux;

  aux = 0.25 * ( a[X] + b[Y] + c[Z] + 1);
  if ( aux > epsilon ) {
    q[W] = sqrt( aux);
    q[X] = 1.0 / ( 4.0 * q[W] ) * ( b[Z] - c[Y] );
    q[Y] = 1.0 / ( 4.0 * q[W] ) * ( c[X] - a[Z] );
    q[Z] = 1.0 / ( 4.0 * q[W] ) * ( a[Y] - b[X] );
  }
  else {
    q[W] = 0.;
    aux = 0.5 * ( b[Y] + c[Z]);
    if ( aux > epsilon ) {
      q[X] = sqrt( aux);
      q[Y] = b[X] / ( 2.0 * q[X] );
      q[Z] = c[X] / ( 2.0 * q[X] );
    }
    else {
      q[X] = 0.0;
      aux = 0.5 * ( 1.0 - c[Z] );
      if ( aux > epsilon ) {
        q[Y] = sqrt( aux);
        q[Z] = b[Z] / ( 2.0 * q[Y] );
      }
```

```
        else {
          q[Y] = 0.0;
          q[Z] = 1.0;
        }
      }
    }
  }
```

Annexe B

Contrôle des splines cubiques

Les splines sont depuis longtemps utilisées en synthèse d'images et en animation. Parmi les nombreuses familles de splines, les splines cubiques par morceaux sont sans doute les plus utilisées. Elles offrent plus de souplesse que les courbes de niveau inférieur et requièrent moins de temps de calcul que celles d'ordre supérieur.

Une spline cubique peut se représenter matriciellement par:

$$P(u) = \begin{pmatrix} u^3 & u^2 & u & 1 \end{pmatrix} M_{spline} \begin{pmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{pmatrix} \qquad (B.1)$$

Ici, le paramètre u varie entre 0 et 1. Les C_i sont les contraintes géométriques qui, selon les types de splines, représentent des tangentes ou des points de contrôle. On note $C = \begin{pmatrix} C_1 & C_2 & C_3 & C_4 \end{pmatrix}^T$; M_{spline} est une matrice 4×4 caractéristique du type de courbe.

B.1 Définition

Les splines d'Hermite admettent comme contraintes deux points de contrôle P_0 et P_1 ainsi que deux tangentes D_0 et G_1. Nous avons donc ici $C = \begin{pmatrix} P_0 & D_0 & P_1 & G_1 \end{pmatrix}$.

P_0 et P_1 définissent les extrémités de la courbe en $u = 0$ et $u = 1$ d'où les équations:

$$P(0) = P_0 = \begin{pmatrix} 0 & 0 & 0 & 1 \end{pmatrix} M_{Hermite} C \qquad (B.2)$$

$$P(1) = P_1 = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} M_{Hermite} C \qquad (B.3)$$

$$(B.4)$$

D_0 est la tangente à droite de la courbe quand $u = 0$ et G_1 est la tangente à gauche quand $u = 1$. Dans la définition habituelle des splines d'Hermite, le paramètre G_1 représente la vitesse pour $u = 1$. C'est un vecteur qui n'est pas du bon côté de la courbe

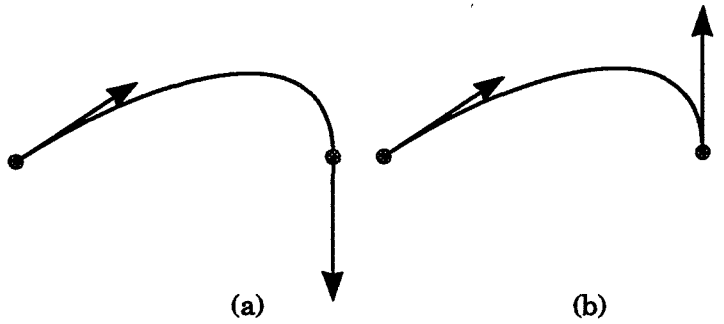


FIG. B.1 Deux façons de voir les tangentes

(figure B.1.a). Nous préférons que la tangente en $u = 1$ soit du même côté que la courbe, ce qui est plus intuitif pour l'utilisateur (figure B.1.b). Pour cela, G_1 représente l'opposé de la vitesse à l'extrémité de la courbe. En dérivant l'équation B.1 pour obtenir l'expression donnant la tangente en u:

$$P'(u) = \begin{pmatrix} 3t^3 & 2t^2 & t & 0 \end{pmatrix} M_{Hermite} C$$

Les contraintes sur les tangentes s'écrivent:

$$P'(0) = D_0 = \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} M_{Hermite} C \quad \text{(B.5)}$$

$$-P'(1) = G_1 = \begin{pmatrix} -3 & -2 & -1 & 0 \end{pmatrix} M_{Hermite} C \quad \text{(B.6)}$$

(B.7)

En combinant les équations (B.2), (B.3, (B.5) et (B.6), nous obtenons:

$$C = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ -3 & -2 & -1 & 0 \end{pmatrix} M_{Hermite} C$$

ce qui nous donne par inversion l'expression de la matrice caractéristique de l'interpolation d'Hermite (par rapport à la matrice habituelle, les éléments de la dernière colonne sont inversés):

$$M_{Hermite} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ -3 & -2 & -1 & 0 \end{pmatrix}^{-1} = \begin{pmatrix} 2 & 1 & -2 & -1 \\ -3 & -2 & 3 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

B.2 Contrôle des tangentes

Grâce aux tangentes qui interviennent directement dans la formulation d'Hermite, le contrôle de ces courbes est très facile. Il suffit de permettre à l'utilisateur de manipuler

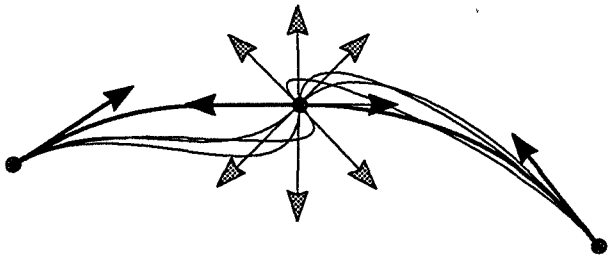


FIG. B.2 Variation de la direction des tangentes

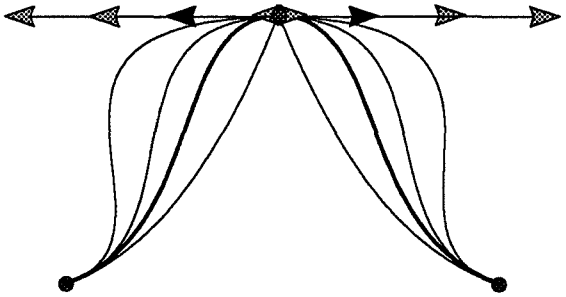


FIG. B.3 Variation du module des tangentes

l'extrémité de ces tangentes. Des facilités peuvent être fournies afin d'assurer certaines conditions de continuité (dans les exemples fournis, la courbe est constituée de deux morceaux de spline) :

- Variations de la direction. Les demi-tangentes gauche et droite restent colinéaires (figure B.2).
- Variations du module. Les demi-tangentes gauche et droite restent colinéaires (figure B.3).
- Variation de la direction de la demi-tangente à droite. Création d'un effet de discontinuité (figure B.4).
- Variations du module de la demi-tangente à droite. La courbe garde la continuité géométrique G_1 mais perd la C_1-continuité (figure B.5).

B.3 Contrôle de la tension, du biais et de la continuité

Dans cette approche introduite par Kochanek et Bartels [KB84], le contrôle se fait par l'intermédiaire de trois paramètres que l'utilisateur peut spécifier à chaque point de contrôle. Ces trois paramètres intuitifs que sont la tension, le biais et la continuité influent

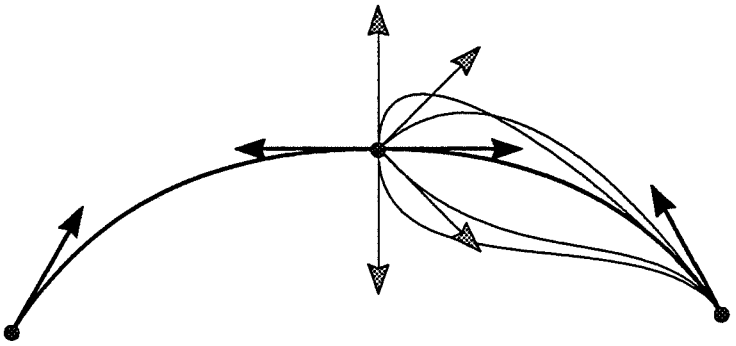


FIG. B.4 Effet de discontinuité en variant la direction de la demi-tangente à droite

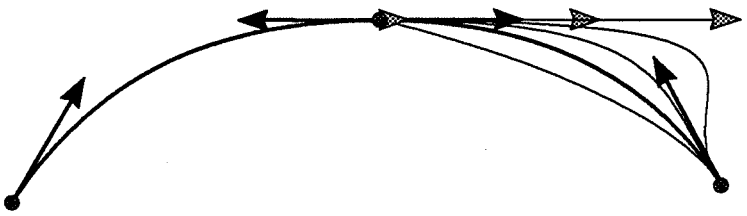


FIG. B.5 Continuité G_1

sur l'expression des tangentes dans la formulation des splines d'Hermite. En un point P_i, les tangentes à gauche et à droite sont données par les formules suivantes :

$$G_i = \frac{1}{2}((1-t)(1-c)(1+b)(P_i - P_{i-1}) + (1-t)(1+c)(1-b)(P_{i+1} - P_i))$$
$$D_i = \frac{1}{2}((1-t)(1+c)(1+b)(P_i - P_{i-1}) + (1-t)(1-c)(1-b)(P_{i+1} - P_i))$$

Ces trois paramètres influent sur l'allure de la courbe :

- t $(-1 \leq t \leq 1)$ permet de modifier la tension de la courbe. Une valeur élevée produit une courbe plus raide, une valeur négative, une courbe plus arrondie (figure B.6).
- b $(-1 \leq t \leq 1)$ permet d'introduire un effet de décalage de la courbe autour du point de contrôle (figure B.7).
- c $(-1 \leq t \leq 1)$ fait varier la continuité de la courbe (figure B.8).

Quand les trois paramètres sont nuls (ce sont les valeurs par défaut), on retrouve la valeur des tangentes des splines de Catmull-Rom [CR74].

D'autres approches autorisent un contrôle précis sur des splines cubiques, en particulier les Beta-splines [BB83] qui sont une extension de l'idée de Kochanek et Bartels. Nous ne les exposerons pas ici.

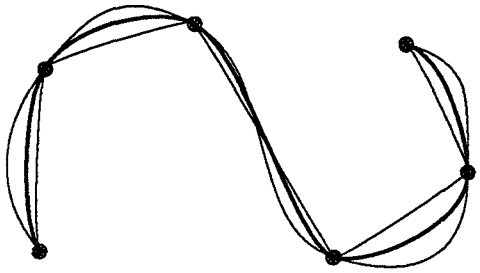


FIG. B.6 Diverses valeurs de tension appliquées à tous les points de la courbe

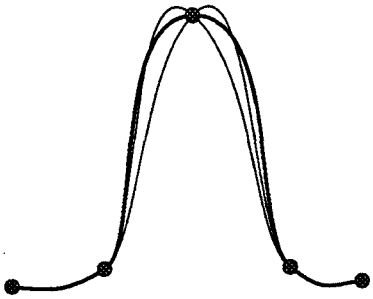


FIG. B.7 Diverses valeurs de biais appliquées au point central de la courbe

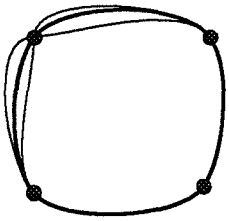


FIG. B.8 Diverses valeurs de continuité appliquées à tous les points de la courbe

Annexe C

Minimisation d'une fonction

Dans cette annexe, nous donnons quelques principes d'optimisation. Etant donné une fonction f dépendant d'une ou plusieurs variables, il s'agit de déterminer les valeurs de ces variables pour lesquelles f admet un minimum. De nombreuses méthodes existent [GMW82]. Celle que nous décrivons suppose que l'on sache calculer les dérivées premières et secondes de la fonction.

Commençons par considérer le problème de la recherche d'un minimum local d'une fonction univariée $f(x)$, sans contraintes. Les conditions d'optimalité (conditions nécessaires et suffisantes à l'obtention d'un tel minimum) sont :

$$f'(x) = 0 \tag{C.1}$$

$$f''(x) > 0 \tag{C.2}$$

Considérons le développement limité de Taylor à l'ordre deux de cette fonction :

$$f(x+\epsilon) = f(x) + \epsilon f'(x) + \frac{1}{2}\epsilon^2 f''(x+\theta\epsilon)$$

Si $f'(x) \neq 0$, il est facile de trouver ϵ tel que $f(x+\epsilon) < f(x)$, auquel cas x n'est pas un minimum. Ceci justifie la condition (C.1). Si la condition d'ordre un, $f'(x) = 0$, est vérifiée, x est un point stationnaire (pas forcément un minimum local[1]) et nous avons :

$$f(x+\epsilon) = f(x) + \frac{1}{2}\epsilon^2 f''(x+\epsilon\theta)$$

Si $f''(x) < 0$, on peut trouver ϵ tel que $f(x+\epsilon) < f(x)$ ce qui justifie la condition (C.2).

De même que dans le cas univarié, les conditions d'optimalité pour une fonction vectorielle $f(x), x \in \mathcal{R}^n$, s'obtiennent à partir du développement en série de f :

$$f(x+p) = f(x) + g^T(x)p + \frac{1}{2}p^T H(x)p \quad p \in \mathcal{R}^n$$

[1] par exemple, l'origine est un point stationnaire pour la fonction $f(x) = x^3$ puisque sa dérivée s'annule, mais ce n'est pas un minimum.

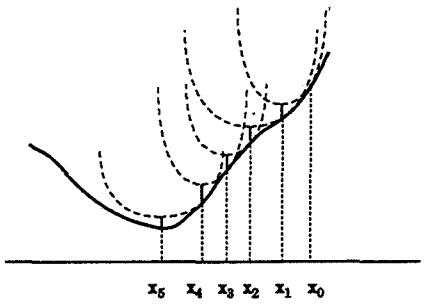


FIG. C.1 Recherche du minimum d'une fonction univariée

où g désigne le gradient de f, c'est a dire le vecteur de $\mathcal{R}^n$ des dérivées partielles et H désigne le hessien de f, c'est à dire la matrice carrée d'ordre n des dérivées partielles d'ordre 2 :

$$g(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

et

$$H(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Les conditions d'optimalité sont obtenues quand la dérivée de f s'annule, c'est à dire :

$$g^T(x) + p^T H(x) = g(x) + H(x)p = 0 \qquad (C.3)$$

La condition d'ordre deux $f'' = 0$ dans le cas univarié revient maintenant à dire que la matrice H est définie positive. En pratique, cette contrainte est implicitement vérifiée par un choix judicieux de la fonction f (f est souvent quadratique).

L'algorithme de minimisation est itératif. A chaque étape, la résolution du système (C.3) fournit un vecteur p minimisant l'approximation de f à l'ordre deux. La nouvelle estimation du minimum sera le vecteur $x = x + p$. Pour une fonction univariée, cela correspond à approximer $f(x)$ par une parabole (figure C.1). A l'étape suivante, on prendra comme nouvelle estimation du minimum le point x représentant le minimum de la parabole.

Liste des figures

Bibliographie

[AA92] Akman (V.) et Arslan (A.). – Sweeping with all graphical ingredients in a topological picturework. *Comput. & Graphics*, vol. 16, n° 3, 1992, pp. 273–281.

[AG85] Armstrong (W.W.) et Green (M.W.). – The dynamic of articulated rigid bodies for purpose of animation. *The Visual Computer*, vol. 1, n° 4, décembre 1985, pp. 231–240.

[AGL87] Armstrong (W.W.), Green (M.W.) et Lake (R.). – Near real time control of human figure models. *IEEE Computer Graphics & Applications*, juin 1987, pp. 52–61.

[Arn88] Arnaldi (B.). – *Conception du Noyau d'un Système d'Animation de Scènes Tridimensionnelles intégrant les lois de la Dynamique.* – Thèse de doctorat, Université de Rennes I, juillet 1988.

[BB83] Beatty (J.C.) et Barsky (B.A.). – Controling the shape of parametric b-spline and beta-spline curves. In : *Graphics Interface'83*, pp. 223–231.

[BB88] Barzel (R.) et Barr (A.). – A modelling system based on dynamic constraints. *Computer Graphics*, vol. 22, n° 4, août 1988, pp. 179–188. – Proc. SIGGRAPH'88.

[BC89] Bruderlin (A.) et Calvert (T.W.). – Goal directed, dynamic animation of human walking. *Computer Graphics*, vol. 23, n° 3, juillet 1989, pp. 233–242. – Proc. SIGGRAPH'89.

[BCGH92] Barr (A.H), Currin (B.), Gabriel (S.) et Hughes (J.F.). – Smooth interpolation of orientations with angular velocity constraints using quaternions. *Computer Graphics*, vol. 26, n° 2, 1992, pp. 313–320. – Proc. SIGGRAPH'92.

[Ber76] Bernhard (P.). – *Commande Optimale, Décentralisation et Jeux Dynamiques.* – Paris, Dunod Automatique, Bordas, 1976.

[BH75] Bryson (A.E.) et Ho (Yu-Chi). – *Applied Optimal Control: Optimization, Estimation and Control.* – Hemisphere Publishing Corporation, 1975.

[BH89] Bartels (R.H.) et Hardtke (I.). – Speed adjustment for key-frame interpolation. In : *Proceedings of Graphics Interface '89*, pp. 14–19.

[BKK+86] Badler (N.I.), Korein (J.D.), Korein (J.U.), Radack (G.M.) et Brotman (L.S.). – Positionning and animating human figures in a task-oriented environment. *The Visual Computer*, vol. 2, 1986, pp. 63–71.

[BMP93] Benouamer (M.), Michelucci (D.) et Peroche (B.). – Boundary evaluation using lazy rationnal arithmetic : A detailed implementation. *2nd ACM, IEEE symposium on Solid Modeling & Applications '93*, mai 1993.

[BMW87] Badler (N.I.), Manoochehri (K.H.) et Walters (G.). – Articulated figure positionning by multiple constraints. *IEEE Computer Graphics & Applications*, June 1987, pp. 28–38.

[BN88] Brotman (L.S.) et Netravali (A.N.). – Motion interpolation by optimal control. *Computer Graphics*, vol. 22, n° 4, août 1988, pp. 309–315. – Proc. SIGGRAPH'88.

[BOK80] Badler (N.I.), O'Rourke (J.) et Kaufman (B.). – Special problems in human mouvment simulation. *Computer Graphics*, vol. 14, 1980, pp. 189–197.

[Bor87] Borgne (M. Le). – *Quaternion et Contrôle sur l'Espace des Rotations.* – Rapport de recherche n° 751, INRIA, novembre 1987.

[BP90] Beigbeder (M.) et Peroche (B.). – *un système de synthèse d'images 3D : Illumines.* – Rapport de recherche n° 90–2, Ecole des Mines de Saint-Etienne, 1990.

[CB88] Cros (F.) et Brock (P.J.). – A method for providing full interactive control of the shape of 3d curves & surfaces. In : *Eurographics'88*, pp. 443–455.

[CH89] Cousin (O.) et Hegron (G.). – Une modélisation des cônes généralisés à l'aide des splines. *CFAO*, vol. 4, n° 3, 1989.

[Coh92] Cohen (M. F.). – Interactive spacetime control for animation. *Computer Graphics*, vol. 26, n° 2, 1992, pp. 293–302. – Proc. SIGGRAPH'92.

[CR74] Catmull (E.) et Rom (R.). – A class of local interpolating splines. In : *Computer Aided Geometric Design*, éd. par Academic Press (Inc.), pp. 317–326.

[DAH89] Dumont (G.), Arnaldi (B.) et Hegron (G.). – Mechanics of solids for computer animation. In : *PIXIM'89*, pp. 293–307.

[DB88] DeRose (T.D.) et Barsky (B.A.). – Geometric continuity, shape parameter, and geometric constructions of catmull-rom splines. *ACM Transactions on Graphics*, vol. 7, n° 1, janvier 1988, pp. 1–41.

[DGV91] Dumont (G.), Gascuel (M.P.) et Verroust (A.). – *Animation Contrôlée par la Dynamique - Etat de l'Art.* – Rapport de recherche n° 571, Rennes, IRISA, février 1991.

[dPFV90] de Panne (M. Van), Fiume (E.) et Vranesic (Z.). – Reusable motion synthesis using state-space controllers. *Computer Graphics*, vol. 24, n° 4, août 1990, pp. 225–234. – Proc. SIGGRAPH'90.

[Duf86] Duff (T.). – Splines in animation and modelling. In : *State of the Art in Image Synthesis.* – SIGGRAPH'86 Courses Notes, number 5.

[Dum90] Dumont (G.). – *Animation de Scènes Tridimensionnelles: la Mécanique des Solides comme Modèle de Synthèse du Mouvement.* – Thèse de doctorat, Université de Rennes I, mai 1990.

[Fea83] Featherstone (R.). – The calculation of robot using articulated-body inertias. *International Journal of Robotics Research*, vol. 2, n° 1, 1983, pp. 13–30.

[FvDFH90] Foley (J.D.), van Dam (A.), Feiner (S.K.) et Hughes (J.F.). – *Computer Graphics, Principles and Practice.* – Addison-Wesley Publishing Company, 1990.

[Gan89] Gançarski (P.). – *Animation Informatisée: Présentation et Etat de l'Art.* – Rapport de recherche, Strasbourg, Université Louis Pasteur, février 1989.

[Gas89] Gascuel (M.P.). – Osea: un nouveau modèle de matière pour traiter les collisions entre objets déformables. In : *PIXIM'89*, pp. 309–323.

[Gas90] Gascuel (M.P.). – *Déformations de surfaces complexes: Techniques de haut niveau pour la modélisation et l'animation.* – Thèse de doctorat, Université de Paris Sud. Centre d'Orsay, octobre 1990.

[Gea91] Gear (C.W.). – *Numerical Initial Value Problems in Ordinary Differential Equations.* – Englewood Cliffs, New Jersey, Prentice Hall, Inc., 1991. Series in Automatic Computation.

[GG92] Gascuel (J.-D.) et Gascuel (M-P). – Displacement constraints: a new method for interactive dynamic animation of articulated bodies. In : *Third Eurographics Workshop on Animation and Simulation.* – Cambridge, 1992.

[GK85] Gabriel (S.) et Kajiya (J.). – Spline interpolation in curved space. In : *State of the Art in Image Synthesis.* – SIGGRAPH'85 Course Notes, 1985.

[GMW82] Gill (P.E.), Murray (W.) et Wright (M.H.). – *Practical Optimization.* – London, Academic Press, Inc., 1982.

[GY89] Gagalowicz (A.) et Yahia (H.). – Conception et implémentation d'un modeleur d'animation. *Bigre*, no61–62, avril 1989, pp. 144–150.

[HAD88] Hégron (G.), Arnaldi (B.) et Dumont (G.). – Toward general animation control. In : *Proceedings of CG international'88*, pp. 54–63.

[Ham44] Hamilton (Sir W. R.). – On quaternions; or on a new system of imaginaries in algebra. *Philosophical Magazine*, noXXV, juillet 1844, pp. 10–13.

[Han91] Hanotaux (G.). – Contrôle interactif d'un solide rigide articulé. In : *Gros Plan 91*, pp. 203–210.

[Han92a] Hanotaux (G.). – Interactive control of orientation interpolations. In : *Third Eurographics Workshop on Animation and Simulation.* – Cambridge, 1992.

[Han92b] Hanotaux (G.). – *Interpolation d'Orientations pour l'Animation.* – Rapport de recherche n° 92–5, Ecole des Mines de Saint-Etienne, avril 1992.

[HM88] Heise (R.) et MacDonald (B.A.). – Quaternion & motion interpolation : a tutorial. In : *Proceedings of CG international'89*, pp. 54–63.

[HP93] Hanotaux (G.) et Peroche (B.). – Interactive control of interpolation for animation and modeling. In : *Graphics Interface'93*, pp. 201–208. – Toronto, Canada, mai 1993.

[IC87] Isaacs (P.M.) et Cohen (M.). – Controlling dynamic simulation with kinematic constraints, behavior functions & inverse dynamic. *Computer Graphics*, vol. 21, n° 4, juillet 1987, pp. 215–224. – Proc. SIGGRAPH'87.

[IC88] Isaacs (P.M.) et Cohen (M.). – Mixed methods for complex kinematic constraints in dynamic figure animation. *The Visual Computer*, vol. 4, 1988, pp. 296–305.

[KB84] Kochanek (D.H.U.) et Bartels (R. H.). – Interpolating splines with local tension, continuity and bias control. *Computer Graphics*, vol. 18, n° 3, juillet 1984, pp. 33–41. – Proc. SIGGRAPH'84.

[KB86] Korein (J.U.) et Badler (N.I.). – Techniques for generating the goal-directed motion of articulated structures. In : *Advanced Computer Animation*, pp. 71–81. - SIGGRAPH'86 Course Notes, 1986.

[Klo86] Klok (F.). – Two moving coordinate frames for sweeping along 3d trajectories. *Computer Geometric Aided Design*, vol. 3, 1986, pp. 217–229.

[Laf75] Lafon (J.C.). – Sur le produit de deux quaternions. *C. R. Acad. Sc. Paris*, 1975. – t. 280, 10 mars 1975, série A - 665.

[Las87] Lasseter (J.). – Principle of traditionnal animation applied to 3d computer animation. *Computer Graphics*, vol. 21, n° 4, juillet 1987, pp. 35–44. – Proc. SIGGRAPH'87.

[LC86] Luciani (A.) et Cadoz (C.). – Utilisation de modèles mécaniques et géométriques pour la synthèse et le contrôle d'images animées. In : *Deuxième Colloque Image CESTA.* – Nice, avril 1986.

[MZ90] McKenna (M.) et Zeltzer (D.). – Dynamic simulation of autonomous legged locomotion. *Computer Graphics*, vol. 24, n° 4, 1990, pp. 29–39. – proc. SIGGRAPH'90.

[Nie91] Nie (Z.G.). – *Utilisation des Déformations pour la Modélisation des Solides de Forme Libre en Synthèse d'Images.* – Thèse de doctorat, Ecole des Mines de Saint-Etienne, octobre 1991.

[NO86] Nielson (G.M.) et Olsen (D.R.). – Direct manipulation techniques for 3d objects using 2d locator devices. In : *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, éd. par Crone (F.) et Pizer (S.M.). pp. 175–182. – Chapell Hill, NC, USA, octobre 1986.

[Ove89] Overveld (C.W.A.M. Van). – Application of a perspective cursor as a 3d locator device. *Computer Aided Design*, vol. 21, n° 10, décembre 1989.

[PF88] Pintado (X.) et Fiume (E.). – Grafields: Field-directed dynamic splines for interactive motion control. In : *Eurographics'88.* pp. 43–54. – Elsevier Science Publishers.

[PFTV92] Press (W.), Flannery (B.), Teukilsky (S.) et Vetterling (W.). – *Numerical Recipes in C: The art of scientific programming, deuxième édition.* – Cambridge, England, Cambridge University Press, 1992.

[Ple88] Pletincks (D.). – The use of quaternions for animation modelling and rendering. In : *Proceedings of CG international'88*, pp. 54–63.

[Ple89] Pletincks (D.). – Quaternion calculus as a basic tool in computer graphics. *The Visual Computer*, vol. 5, 1989, pp. 2–13.

[RA90] Reyes-Avila (L.). – *Quaternion : Une représentation paramétrique systématique des rotations finies.* – Rapport de recherche n° 1303, INRIA, octobre 1990.

[Rip91] Ripp (R.). – *Animation Graphique et Interactivité.* – Strasbourg, Thèse de doctorat, Université Louis Pasteur, mai 1991. Thèse de Doctorat.

[RM88] Rao (K.) et Medioni (G.). – Useful geometric properties of the generalized cone. *Proc. Computer Vision, Graphics and Image Processing*, vol. 27, 1988, pp. 129–156.

[Roe93] Roelens (M.). – *Un environnement pour le tracé de rayon utilisant une modélisation par arbre de construction.* – Thèse de doctorat, Ecole des Mines de Saint-Etienne, avril 1993.

[SB85] Steketee (S.N.) et Badler (N.I.). – Parametric keyframe interpolation incorporating kinetic adjustement and phrasing control. *Computer Graphics*, vol. 19, n° 3, 1985, pp. 255–262. – Proc. SIGGRAPH'85.

[SD91a] Slater (M.) et Davison (A.). – Liberation from flatland : 3d interaction based on the desktop bat. In : *Eurographics'91*, pp. 209–221.

[SD91b] Smith (J.) et Drewery (K.). – Timewarps : A temporal reparametrization paradigm for parametric animation. In : *Proceedings of Eurographics '91*, éd. par Post (F.H.) et Barth (W.). pp. 413–423. – North Holland, septembre 1991.

[Sho85] Shoemake (K.). – Animating rotation with quaternion curves. *Computer Graphics*, vol. 19, n° 3, 1985, pp. 245–254. – Proc. SIGGRAPH'85.

[SW92] Siltanen (P.) et Woodward (C.). – Normal orientation methods for 3d offset curves, sweep surfaces and skinning. *Eurographics'92 proceedings*, vol. 11, n° 3, 1992, pp. 449–457.

[SZ90] Schröder (P.) et Zeltzer (D.). – The virtual erector set : Dynamic simulation with linear recursive constraint propagation. *Computer Graphics*, mai 1990, pp. 23–31.

[vdPFV92] van de Panne (M), Fiume (E.) et Vranesic (Z.G). – Control techniques for physically-based animation. In : *Third Eurographics Workshop on Animation and Simulation.* – Cambridge, 1992.

[Ver89] Verroust (A.). – Animation d'objets articulés à l'aide de la dynamique : Etat de l'art. *Bigre*, no61–62, avril 1989, pp. 151–159.

[vO91] van Overweld (C.W.A.W). – Building blocks for goal directed motion. In : *Second Eurographics Workshop on Animation and Simulation*, pp. 41–54. – Vienne, septembre 1991.

[WB85] Wilhelms (J.) et Barsky (B.). – Using dynamic analysis to animate articulated bodies such as human & robots. In : *Graphics Interface'85*, pp. 197–204.

[Wil87] Wilhelms (J.). – Using dynamic analysis for realistic animation of articulated bodies. *IEEE Computer Graphics & Applications*, juillet 1987, pp. 12–27.

[Wit77] Wittenburg (J.). – *Dynamics of Systems of Rigid Bodies.* – Stuttgart, B.G. Teubner, 1977.

[WK88] Witkin (A.) et Kass (M.). – Spacetime constraints. *Computer Graphics*, vol. 22, n° 4, août 1988, pp. 159–168. – Proc. SIGGRAPH'88.

[WW90] Witkin (A.) et Welch (W.). – Fast animation and control of nonrigid structures. *Computer Graphics*, vol. 24, n° 4, 1990, pp. 243–252. – proc. SIGGRAPH'90.

[YG89] Yahia (H.) et Gagalowicz (A.). – Interactive animation of object orientations. In : *PIXIM'89*, pp. 265–275.

TECHNIQUES DE CONTRÔLE DU MOUVEMENT POUR L'ANIMATION

Gabriel HANOTAUX

mots-clés
synthèse d'images, animation, contrôle du mouvement, interpolation, interface 3D, méthodes d'optimisation, simulation

Résumé
En animation tridimensionnelle, on distingue principalement deux grandes approches basées soit sur des modèles descriptifs, soit sur des modèles générateurs. Dans la première de ces approches, les possibilités de contrôle sur les trajectoires d'interpolation sont essentielles. Je décris une méthode offrant à l'utilisateur les mêmes possibilités d'interaction sur les trajectoires d'orientations que sur les positions. Le contrôle en temps réel des orientations est rendu possible par une paramétrisation à base de logarithmes et d'exponentielles de quaternions et par la notion de tangente sphérique. Une interface 3D de haut niveau gère les interactions avec l'utilisateur. La vitesse de déplacement le long des trajectoire est déterminée par une paramétrisation automatique reprenant des principes de mécanique élémentaire. Enfin, ces techniques sont appliquées à la modélisation interactive de cylindres généralisés.

Les systèmes par paramètres-clés montrent vite leur limitation dès qu'il s'agit d'animer de façon réaliste des objets complexes. Pour cela, les systèmes d'animation - dits générateurs - intégrant les lois de la mécanique ont été introduits. Dans la deuxième partie, je propose une approche alliant les possibilités de contrôle des systèmes descriptifs aux réalisme des modèles générateurs. Le principe est de minimiser l'énergie au cours du mouvement. Les équations du mouvement pour des solides rigides articulés sont construites sous forme symbolique. L'étape de minimisation est réalisée par un algorithme de contrôle optimal, traitant les équations du mouvement sous leur forme continue.

Abstract
In three dimensional animation, we mainly distinguish two important approaches, based either on descriptive or dynamic models. In the first one, the control of interpolating trajectories is essential. I describe a method giving the user the same interactive possibilities for orientation trajectories as for positions. Real-time control of orientations is made possible due to a parametrisation based on the logarithm and the exponential of quaternions and due to the notion of spherical tangent. A high level 3D interface manages user interactions. Motion dynamics is defined by an automatic parametrisation taking elementary mechanical laws into account. Finally, these techniques are applied to the interactive modeling of sweep objects.

Key-frame animation systems are not well suited to the realistic animation of complex objects. With this objective, animation systems based on dynamic laws have been introduced. I propose a method offering the control possibilities of key-frame animation systems and the realism of dynamic models. The principle is to minimize energy along the path. The motion equations for rigid articulated bodies are built in a symbolic form. The minimization step is performed with an optimal control algorithm, treating the motion equations in their continuous form.